



Data Warehouses

M2 D&K, 2017

Benoît Groz
benoit.groz@lri.fr

Univ. Paris-Sud

Outline

M2 D&K 2017

1 Administrativia

- Course overview
- References

Outline

M2 D&K 2017

1 Administrativia

- Course overview
- References

Schedule (tentative)

Hours: Tuesdays 9.00-12.00

Roughly 50% lectures , 50% exercises or lab .

Exam: 15/11, 9.00-11.30 .

Evaluation: $(3 * \text{Exam} + CC)/4$.

CC=lab exam if it can be organized reasonably well. Otherwise, a small exam.



If you fail: the 2nd round Exam replaces all.

What documents may you bring at exam? None!

Prerequisites

Databases is a pre-requisite: bases in SQL (SPJ queries with aggregates,...), relational algebra, ER model.

More advanced database features (indexes, partitioning, etc.) are not a prerequisite and will be covered in this course.

(may be easier if you know a bit on query optimization and join algorithms).

Skills/Notions you will develop:

Challenges of DW.

Principles, models (outdated?).

SQL Aggregates.

+ traditional DB approaches.

Color code

Definition

...

Quizz

...

Example

...

Code

```
SELECT sum(sales.quantitysold)
```

Query result

...

Participation

Please participate. In particular, tell me if

- you do not understand (ask questions)
- I made a mistake
- you know complementary information that your colleagues/I might find useful
- ...

you can also mail me (or try the phone)

QUESTIONS ARE WELCOME !

... I'll do my best to provide adequate answers



Outline

M2 D&K 2017

1 Administrativia

- Course overview
- References

References

This course relies heavily on material from M.Herschel and additional courses or docs mentioned on slides.

Books:

- *Multidimensional databases and Data Warehousing*
C.S. Jensen, T.B.Pedersen and C.Thomsen
- *Data Warehouse Systems, design and implementation*
A.Vaisman, E.Zimanyi
- *The Data Warehouse Toolkit*
R.Kimball, M.Ross (B.U.Orsay)
- *The Data Warehouse Lifecycle Toolkit*
R.Kimball et al.



Survey articles

- *Multidimensional database technology*
T.B.Pedersen, C.S. Jensen
- *An overview of Data Warehousing and OLAP Technology*
S.Chaudhuri, U.Dayal

References (2)

Data Warehouse Courses:

-  *Andreas Geppert* University of Zürich, Crédit Suisse
-  *Wolf-Tilo Balke, Kinda El Maarry* TU Braunschweig
-  *Patrick Marcel* Université de Tour
-  *A. Buckenhofer* DHBW
-  *AC.Caron & Laetitia Jourdan* Université Lille1
-  *Bernard Espinasse* Université Aix-Marseille
-  *Ulf Leser* Humboldt Universität zu Berlin
-  *Erhard Rahm* Universität Leipzig

Useful resources on the web: TDWI, Gartner, video tutorials, vendors doc
(*Oracle*, IBM, Microsoft)...

Lecture content (tentative)

- 1 Administrivia
 - Course overview
 - References
- 2 Data Warehouses: motivations, definition
 - Motivations
 - Definitions: DW, OLTP vs OLAP
 - DW industrial landscape
- 3 DW concepts: the multi-dimensional model
 - Components overview
 - Multidimensional data model: facts, measures, dimensions, cube
 - Multi-dimensional query concepts: OLAP operations
- 4 Relational implementation: schemas
 - Schemas
 - Alternative approaches to store the cube
 - Star and snowflake schemas
 - Additional schema variants
 - Relational implementation: dimensional modeling
 - Fact table
 - Dimension hierarchies
 - Dimension attributes
 - Dimension variations
 - Dimension implementation in Oracle
- 5 Querying the DW: ROLAP technology
 - SQL Recursive queries
 - WITH
 - SQL (OLAP) GROUP BY Extensions
 - Motivation
 - CUBE
 - ROLLUP
 - GROUPING SETS
 - Combinations
 - GROUPING functions
 - GROUPING
 - GROUP_ID
 - GROUP_ID
 - Analytical functions
 - Windows

- 6 Ranking and Aggregation function
 - PIVOT
- 7 Physical design and query optimization
 - Partitioning
 - Partitioning concepts
 - Vertical partitioning
 - Horizontal partitioning: creating partitions
 - Horizontal partitioning: maintenance
 - Horizontal partitioning: benefits
 - Indexes
 - Indexing
 - B-trees
 - Bitmap Indexes
 - Bitmap Join Indexes
 - Joins and partitioning
 - Clustering
- 8 Query optimization: Views
 - Query rewriting
 - View update
 - Materialized Views: support by commercial RDBMS
- 9 Building the DW
 - Architecture
 - Requirements and reference architecture
 - Metadata
 - Data integration: the ETL process
 - Load
- 10 In-memory column stores
 - Introduction to in-memory databases (insp. by H. Plattner's lectures)
 - In-memory databases: commercial systems: MariaDB
 - In-memory databases: commercial systems: Oracle
 - In-memory features
 - IM column store
 - In-memory databases: commercial systems: IBM DB2
 - In-memory features: DB2 12 for z/OS
 - In-memory Column store: DB2 BLU
 - Techniques for handling updates
- 11 Regular expressions

Outline

M2 D&K 2017

② Data Warehouses: motivations, definition

- Motivations
- Definitions: DW, OLTP vs OLAP
- DW industrial landscape

What is a DataWarehouse?

widespread feeling that "society is data rich but information poor"

Business intelligence (BI): set of techniques and tools that enable a company to transform business data into meaningful and useful information for decision making.

DataWarehouse (DW): repository that stores the data and infrastructure to support analysis.

according to R.Kimball:

"a copy of transaction data specifically structured for query and analysis"



cf also Bill Inmon's more precise definition (later).

Outline

M2 D&K 2017

② Data Warehouses: motivations, definition

- Motivations
- Definitions: DW, OLTP vs OLAP
- DW industrial landscape

Motivation story

(source: <http://philip.greenspun.com/sql/data-warehousing.html>)

Let's *imagine* a conversation back in the 90s, between the Chief Information Officer of WalMart and a sales guy from Sybase.

Walmart: "*I want to keep track of sales in all of my stores simultaneously.*"

Sybase: "*You need our wonderful RDBMS software. You can stuff data in as sales are rung up at cash registers and simultaneously query data out right here in your office. That's the beauty of concurrency control.*"

So Walmart buys a \$1 million HP multi-CPU server and a \$500,000 Sybase license, and builds a normalized database:

Sales(product id, store id, quantity sold, date/time of sale)

Products(product id, product name, product category, manufacturer id)

Stores(store id city id, store location, phone number)

Cities(city id, city name, state, population)

Motivation story (continued)

Some time after, a Walmart executive asks: *"I noticed that there was a Colgate promotion recently, directed at people who live in small towns. How much Colgate toothpaste did we sell in those towns yesterday? And how much on the same day a month ago?"*

Her query looks like:

```
SELECT sum(sales.quantitysold)
FROM sales, products, stores, cities
WHERE products.manufacturer_id = 68 -- restrict to Colgate-Palmolive
  and products.product_category = 'toothpaste'
  and cities.population < 40000
  and sales.datetime_of_sale::date = 'yesterday'::date -- restrict to yesterday
  and sales.product_id = products.product_id
  and sales.store_id = stores.store_id
  and stores.city_id = cities.city_id
```

The query returns after 20mins. But dbadmins realize cash registers cannot process the sales when the toothpaste query is run.

Motivation story (continued)

Some time after, a Walmart executive asks: *"I noticed that there was a Colgate promotion recently, directed at people who live in small towns. How much Colgate toothpaste did we sell in those towns yesterday? And how much on the same day a month ago?"*

Her query looks like:

```
SELECT sum(sales.quantitysold)
FROM sales, products, stores, cities
WHERE products.manufacturer_id = 68 -- restrict to Colgate-Palmolive
  and products.product_category = 'toothpaste'
  and cities.population < 40000
  and sales.datetime_of_sale::date = 'yesterday'::date -- restrict to yesterday
  and sales.product_id = products.product_id
  and sales.store_id = stores.store_id
  and stores.city_id = cities.city_id
```

The query returns after 20mins. But dbadmins realize cash registers cannot process the sales when the toothpaste query is run.

Motivation story (continued)

Walmart: *"We type in the toothpaste query and our system wedges."*

Sybase: *"Of course it does! You built an on-line transaction processing (OLTP) system. You can't feed it a decision support system (DSS) query and expect things to work!"*

Walmart: *"But I thought the whole point of SQL and your RDBMS was that users could query and insert simultaneously."*

Sybase: *"Uh, not exactly. The system prevents simultaneous Writes and Reads to guarantee coherent information: this is called "pessimistic locking".*

Walmart: *"Can you fix your system so that it doesn't lock up?"*

Sybase: *"No. But we made this great loader tool so that you can copy everything from your OLTP system into a separate DSS system at 100 GB/hour."*

This new system is the *data warehouse*. This replication of data will

- allow sales to be recorded in the database without interruption
- allow to reorganize the copied data to optimize analytical queries (e.g., better schema)
- allow to integrate smoothly data from different stores (e.g., after a merger with Kmart if Kmart records transactions in a Teradata system).

Motivation story (continued)

Walmart: "We type in the toothpaste query and our system wedges."

Sybase: "Of course it does! You built an on-line transaction processing (OLTP) system. You can't feed it a decision support system (DSS) query and expect things to work!"

Walmart: "But I thought the whole point of SQL and your RDBMS was that users could query and insert simultaneously."

Sybase: "Uh, not exactly. The system prevents simultaneous Writes and Reads to guarantee coherent information: this is called "pessimistic locking".

Walmart: "Can you fix your system so that it doesn't lock up?"

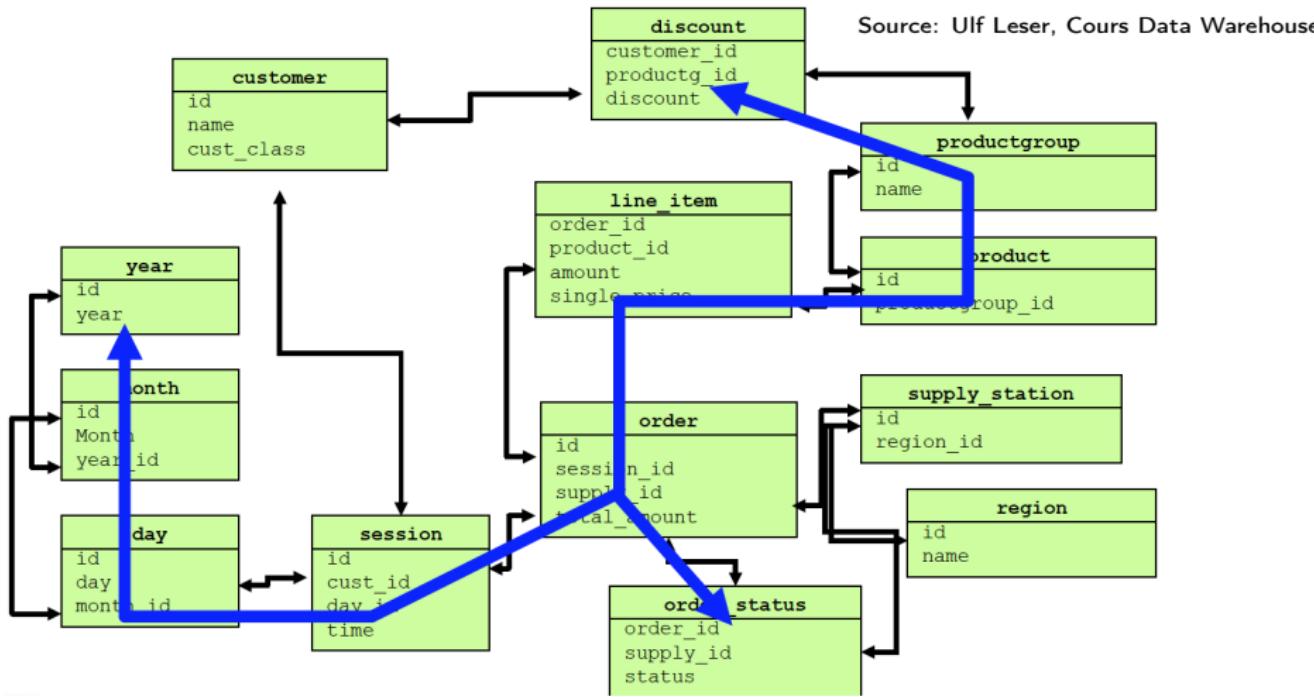
Sybase: "No. But we made this great loader tool so that you can copy everything from your OLTP system into a separate DSS system at 100 GB/hour."

This new system is the *data warehouse*. This replication of data will

- allow sales to be recorded in the database without interruption
- allow to reorganize the copied data to optimize analytical queries (e.g., better schema)
- allow to integrate smoothly data from different stores (e.g., after a merger with Kmart if Kmart records transactions in a Teradata system).

Motivations (2)

Analyst: *"How many sales completed in dec. before Christmas per group of product and discount?"*



Large relations (millions of orders,sessions), many joins \Rightarrow hard query.

Motivations (2)

Analyst: *"How many sales completed in dec. before Christmas per group of product and discount?"*

```
SELECT Y.year, PG.name, DI.disc, count(*)  
FROM year Y, month M, day D, session S,  
line_item I, order O, product P, productgroup PG,  
discount DI, order_status OS  
WHERE M.year_id = Y.id and  
D.month_id = M.id and  
S.day_id = D.id and  
O.session_id = S.id and  
I.order_id = O.id and  
I.product_id = P.id and  
P.productgroup_id = PG.id and  
DI.productgroup_id = PG.id and  
O.id = OS.order_id and  
D.day < 24 and  
M.month = 12  
and OS.status='FINISHED'  
GROUP BY Y.year, PG.name, DI.discount  
ORDER BY Y.year, DI.discount
```

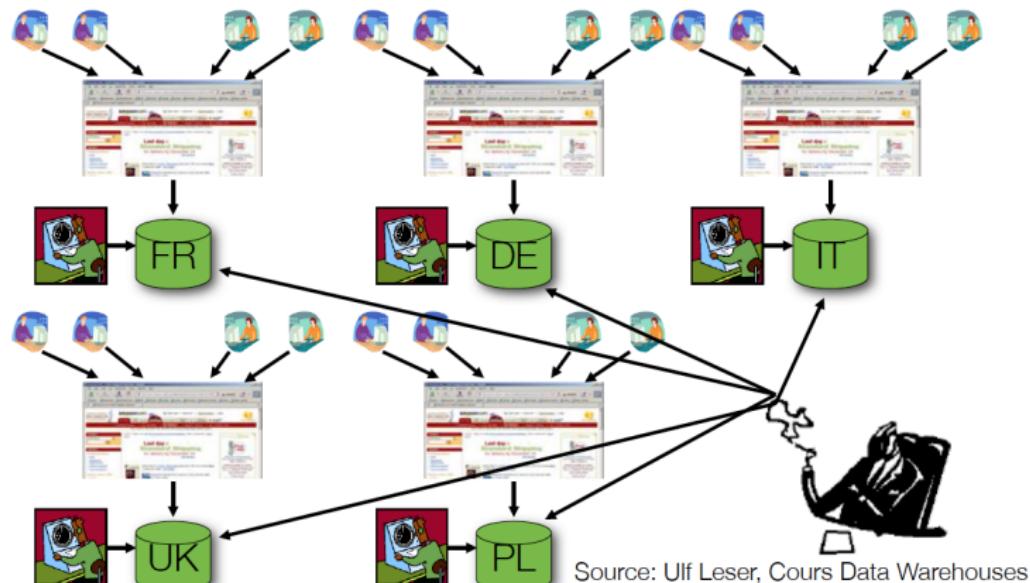
Source: Ulf Leser, Cours Data Warehouses

Large relations (millions of orders,sessions), many joins ⇒ hard query.

Motivations (2)

Analyst: *"How many sales issued in dec. before Christmas per group of product and discount?"*

Amazon.fr, Amazon.de, ...



Source: Ulf Leser, Cours Data Warehouses

Motivations (2)

Analyst: *"How many sales issued in dec. before Christmas per group of product and discount?"*

Amazon.fr, Amazon.de, ...

1. Heterogeneity

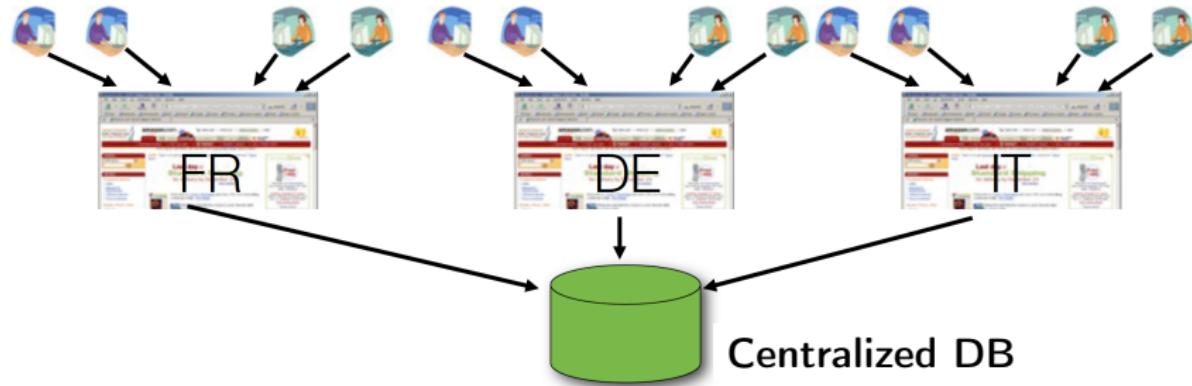
- the database schemas are modified from time to time
- some properties specific to each country (TVA, expedition cost, etc.)
- different semantics for data (measures, etc.)

2. Amount of data

- hard query \Rightarrow can re-use for similar queries if view "christmasOrders"
- Network usage: transfer large amounts of data through web
- Diverging requirements:
 - Operations do not need historical data (purge past orders)
 - The analyst has no need for details (customer name, supplier...)

Motivations (2)

Limitations of some possible solutions

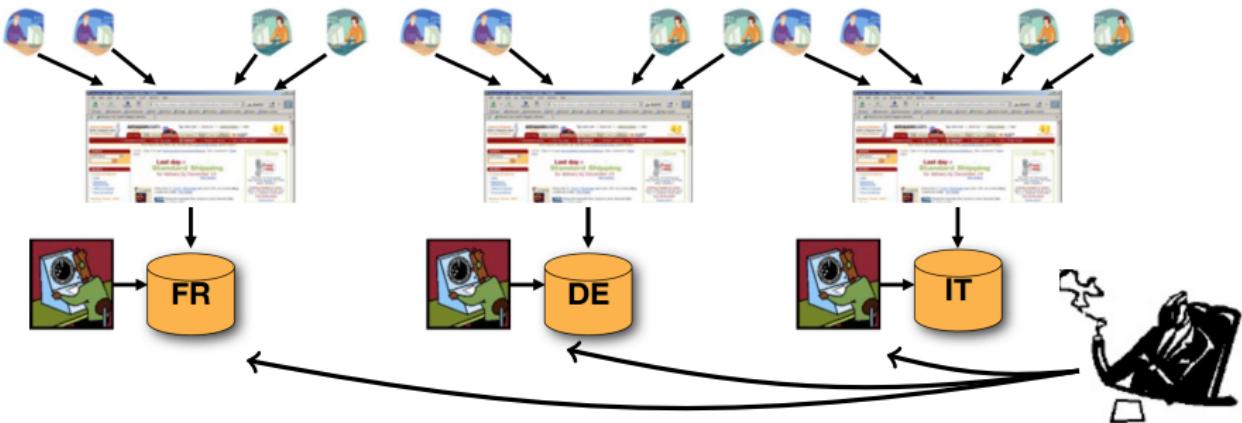


Solution to heterogeneity but

- each branch must connect through the network
- long delay for operations
- does not help with amount of data

Motivations (2)

Limitations of some possible solutions

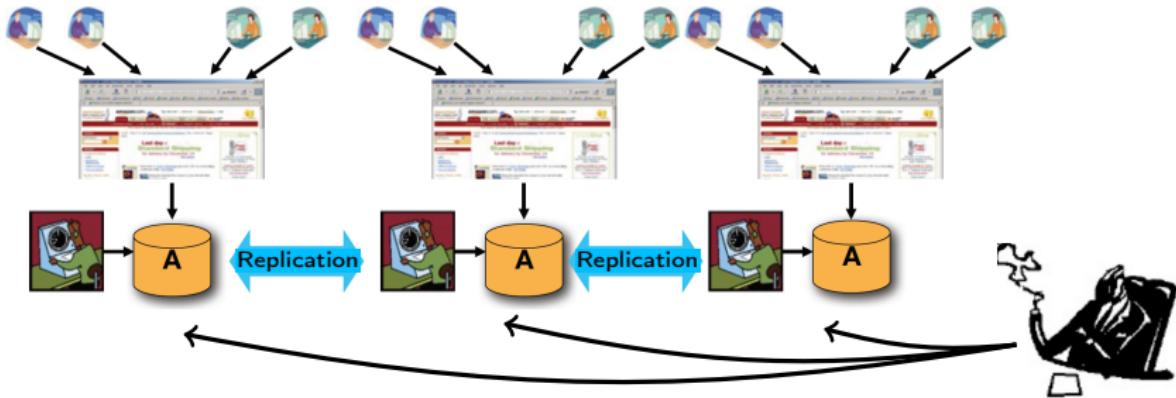


Short delay on operational transactions but

- does not help with heterogeneity issue
- long delay for analytical queries

Motivations (2)

Limitations of some possible solutions

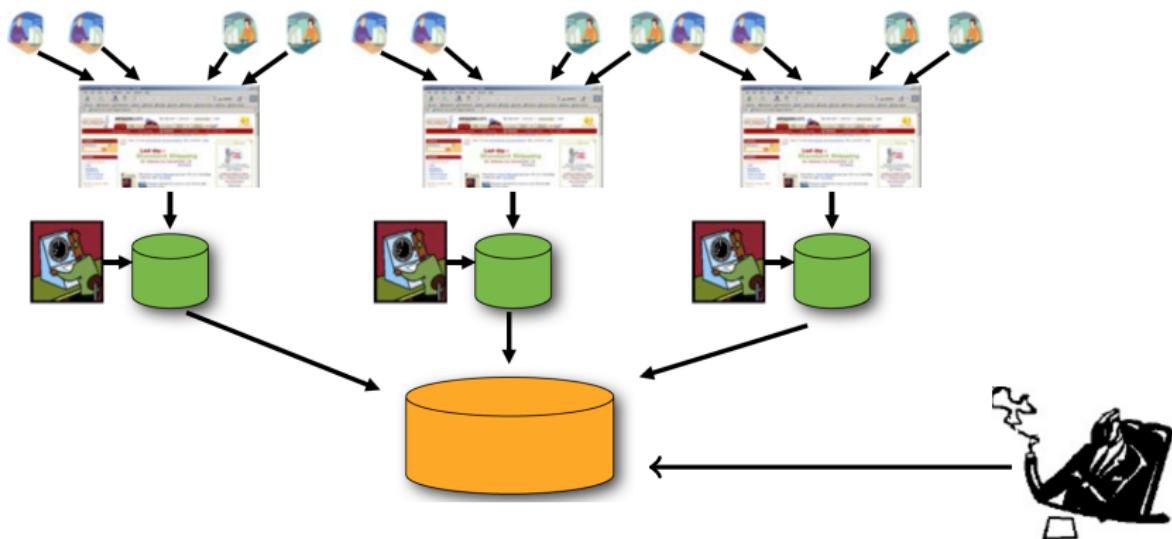


Short delay on analytical queries but

- huge relations in local databases
- longer delay on operational transactions

Motivations (2)

The DataWarehouse approach



- redundant data
- DW stores selected and transformed data
- specific modelization
- asynchronous update of the DW data
- set of specific tools (data preparation, data visualization)

Outline

M2 D&K 2017

② Data Warehouses: motivations, definition

- Motivations
- Definitions: DW, OLTP vs OLAP
- DW industrial landscape

Definition

William H.Inmon's definition ['92]

A data warehouse is a *subject oriented, integrated, time varying, non-volatile* collection of data in support of management's decision making process.



- **subject oriented:** the DW organized according to one or several subjects determined by the analysts' requirements
- **integrated:** the content results from the integration of data from multiple sources
- **time-varying:** keeps track of data changes so that reports show evolution over time
- **non-volatile:** new data can be added, but data is ±never deleted nor updated

OLTP vs OLAP

OLTP: *Online Transaction Processing*

OLAP: *Online Analytical Processing*

Aspect	Operational DB	DW
User	clerk	manager
Concurrency	huge (thousands)	limited (hundreds)
Interaction	short (s)	long analyses (min,h)
Type of interaction	Insert, Update, Delete	Read,periodically (bulk) inserts
Type of query	many simple queries	few, but complex queries (typically drill-down, slice...)
Query scope	a few tuples (often 1)	many tuples (range queries)
Data source	single DB	multiple independant DB...
Schema	query-independant (3NF)	based on queries
Data	original, detailed, dynamic	derived,consolidated, integrated,historicized,partially aggregated,stable
Size	MB,GB	TB,PB
Availability	crucial	not so crucial
Architecture	3-tier (ANSI-SPARC)	adapted to data integration

Outline

M2 D&K 2017

② Data Warehouses: motivations, definition

- Motivations
- Definitions: DW, OLTP vs OLAP
- DW industrial landscape

Application domains

- Retail
- e-business
- Banks, finance
- Insurances
- Telecoms
- Logistics, Travels, Hotels
- Health
- (Life,...) Science
- Public Administrations
- ...

Application: Retail(1)

Walmart

Pioneer: one of the largest (retail) warehouses since. Teradata solution.

	92	2001	2004	2008
size	1 TB	70 TB	>500 TB	2.5 PB

Innovative practices... and secretive.

\$20M Prototype DW to analyze sales history launched in 1990 while still unstable (40% queries rejected). A study over selected analysts estimated ROI per query at \$12,000.

[*Data Warehousing: Using the Wal-Mart Model*, Westman]

I'll spare you the legendary data mining story about "beer and diapers" ;)

[<http://www.dssresources.com/newsletters/66.php>]

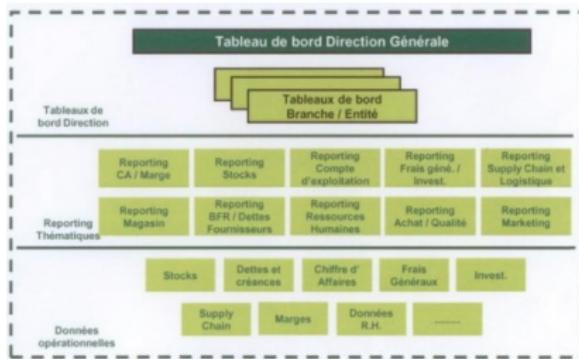
Application: Retail(2)

Casino group

One of the earliest DW in France. Teradata solution

	94	2002	2009
size	80 GB	10 TB	18 TB
users	50	1,500	3500
queries/day		25,000	600,000

*Saved millions when realized
that Coca-Cola stocks were often low [Marcel][Espinasse]*



Sources: <http://www.mycustomer.com/topic/technology/casino-group-upgrades-teradata-data-warehouse>
<http://fr.teradata.com/newsrelease.aspx?id=12338>
<http://www.lemagit.fr/actualites/2240197570/>
Pour harmoniser ses calculs de marge Casino s'engage dans la refonte de son décisionnel

Application: Retail(3)

Goals:

- improve sales, product offerings
- optimize supply chain
- merchandising (store layouts. . .)
- optimize promotions
- customer retention
- compliance

Application: Telecom

Bouygues Telecom

*Back in 2007: due to market saturation, crucial shift from acquiring new customers to hanging on to the really valuable ones. Bouygues had disintegrated architecture with 3-4 data warehouses and 300 data marts
⇒ BI results were unreliable.*

Restructured into a single Teradata DW:

1. consolidate analytics into a single DW (CRM, call detail records)
2. shorten latency: closer to real-time
3. support temporary "sandbox" access to the DW
4. linear scalability

E.g. compute customer churn scores in 4h.

Reduced maintenance overhead ⇒ 33% cost savings for the DW.

[<http://assets.teradata.com/resourceCenter/downloads/CaseStudies/EB6365.pdf>

Sandbox: an independent part of the DWH where people can experiment with new data (check quality) and tools.]

Application: Telecom (2)

France Telecom

Situation in 2002: Consolidates 50 Databases, 80 TB capacity, migration from hourly update to continuous.

Oracle solution on HP machine.

Performances: 500M CDR/day (100GB). 180 Billion CDR stored. 8000 end users, 600 peak concurrent access.

4s for standard queries.

[<http://www.oracle.com/technetwork/testcontent/vldw-cases-winter-132893.pdf>]

Goal summary for Telecom:

- Analyze traffic
- understand customer behavior (churn, product lifecycle, profiling)
- finance/billing operations

Other typical goals (Finance, Insurance, Health)

- risk management
- claims analysis
- asset& liability management
- compliance

Product (suppliers)

Commercial:

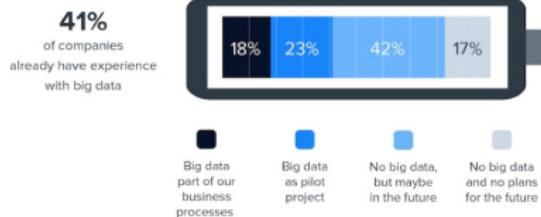


Open Source:



600 M US \$ acq. 06/2015

"Big data" BI market



But there are big regional and industry-specific differences.



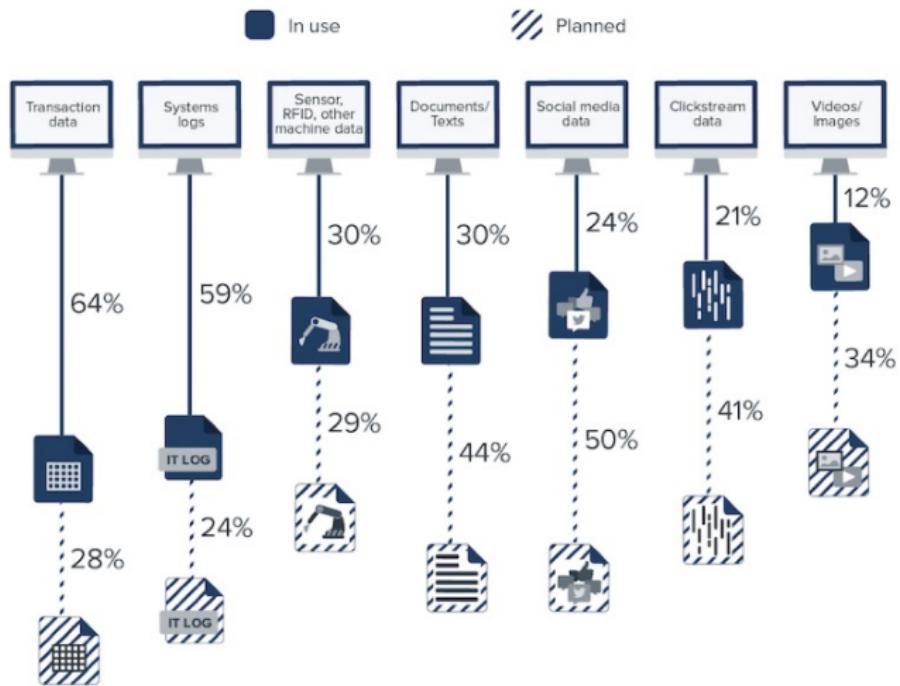
[<http://barc-research.com/big-data-use-cases-2015-infographic/>]

2015 survey conducted over 550 firms by Barc research: market analysis firm sponsored by hp, teradata, tableau, cloudera, etc.

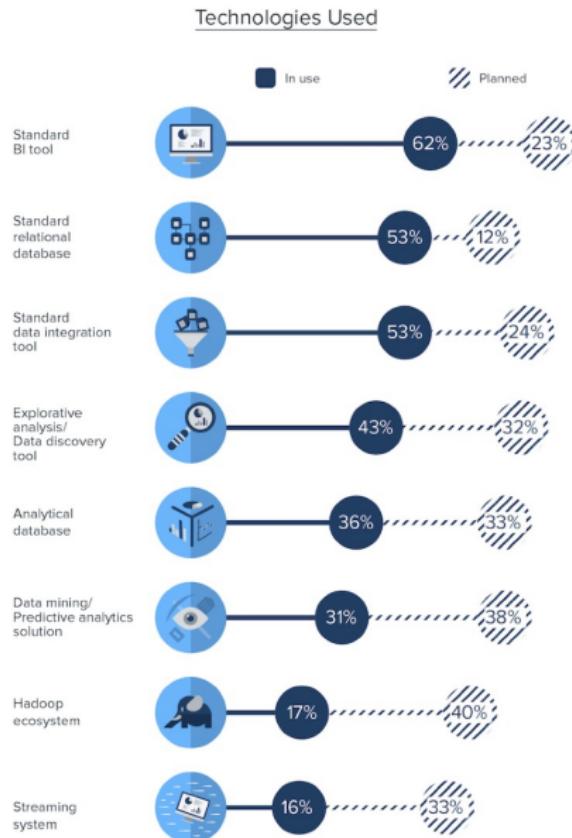
"Big data" evolution



Types of Data Analyzed



Practices are evolving quickly



These lectures focus on traditional SQL-based architectures (no Cloud. See Manolescu and Biffet).

Outline

M2 D&K 2017

③ DW concepts: the multi-dimensional model

- Components overview
- Multidimensional data model: facts, measures, dimensions, cube
- Multi-dimensional query concepts: OLAP operations

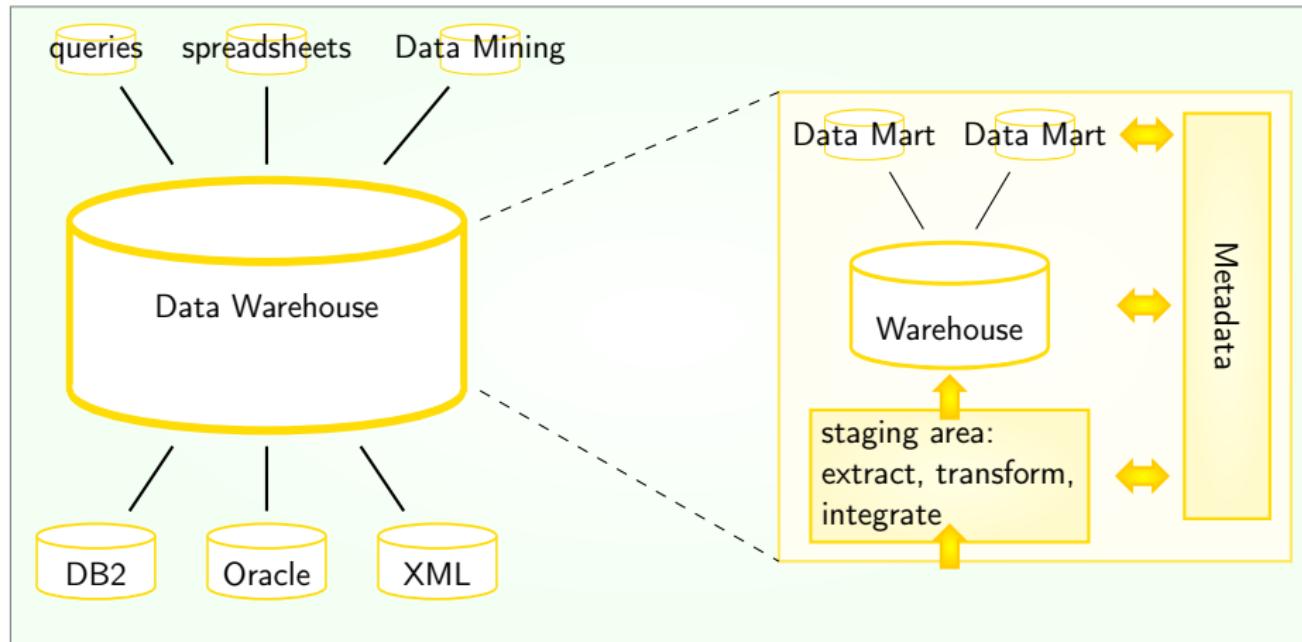
Outline

M2 D&K 2017

③ DW concepts: the multi-dimensional model

- Components overview
- Multidimensional data model: facts, measures, dimensions, cube
- Multi-dimensional query concepts: OLAP operations

DW architecture



Birds' view on DW architecture

Outline

M2 D&K 2017

③ DW concepts: the multi-dimensional model

- Components overview
- Multidimensional data model: facts, measures, dimensions, cube
- Multi-dimensional query concepts: OLAP operations

Why a new model?

- 3NF relational schema too complex for BI queries
- ... and suffer(ed?) from slow query performance

Multi-dimensional model

- is easy to understand for Business users
(OLAP exploration, generalizes spreadsheets. . .)
- delivers fast query performance
- schemas will not need to be reorganized too much over time

From Spreadsheet to the Cube

Sales per period and product in Paris.

	Beverages	Produce	Condiments	Seafood
Q1	21	10	18	35
Q2	27	14	11	30
Q3	26	12	35	32
Q4	14	20	47	31

Product dimension

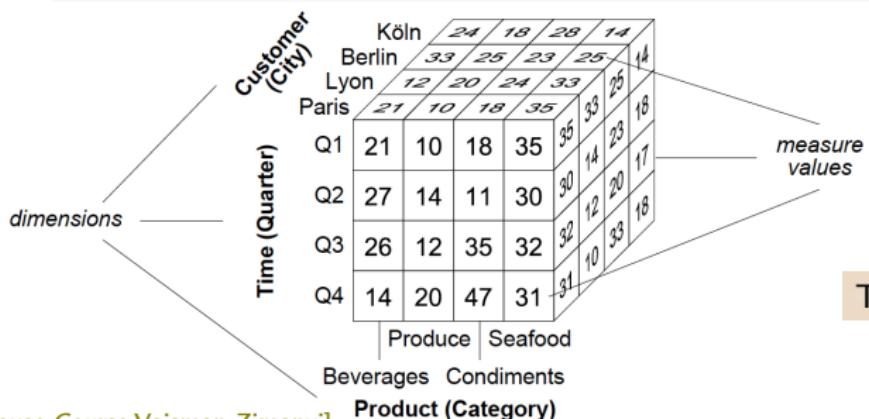
Time dimension

From Spreadsheet to the Cube

	Beverages	Produce	Condiments	Seafood	Köln	3rd dimension = location
Q1	24	18	28	14		
Q2	Beverages	Produce	Condiments	Seafood	Berlin	
Q3	Q1	33	25	23	25	Lyon
Q4	Q2	Beverages	Produce	Condiments	Seafood	
Q3	Q1	12	20	24	33	
Q4	Q2	Beverages	Produce	Condiments	Seafood	Paris
Q3	Q1	21	10	18	35	
Q4	Q2	27	14	11	30	
Q3	Q3	26	12	35	32	
Q4	Q4	14	20	47	31	

From Spreadsheet to the Cube

	Beverages	Produce	Condiments	Seafood	Köln
Q1	24	18	28	14	
Q2	Beverages	Produce	Condiments	Seafood	Berlin
Q3	Q1	33	25	23	
Q4	Q2	Beverages	Produce	Condiments	Seafood
	Q1	12	20	24	33
	Q2	Beverages	Produce	Condiments	Seafood
	Q1	21	10	18	35
	Q2	27	14	11	30
	Q3	26	12	35	32
	Q4	14	20	47	31



The multidimensional model

Data viewed as n dimensional cube (here $n = 3$).

Associated to the cube are:

- Dimension: perspective used to analyze the data
- Cell: the intersection of dimension values
- Fact: non-empty cell
- Measure: numeric value of the cells

The diagram illustrates a 3D cube representing multidimensional data. The cube is defined by four dimensions: Customer (City) on the top face, Time (Quarter) on the left face, Product (Category) at the bottom, and Measure values along the right edge. The Customer dimension has four cities: Köln, Berlin, Lyon, and Paris. The Time dimension has four quarters: Q1, Q2, Q3, and Q4. The Product dimension has four categories: Produce, Seafood, Beverages, and Condiments. The Measure values are represented by the numerical values in the cells of the cube, such as 24, 18, 28, 14 for Köln in Q1.

				Köln	24	18	28	14	
				Berlin	33	25	23	25	14
				Lyon	12	20	24	33	25
				Paris	21	10	18	35	18
			Q1	21	10	18	35	35	23
			Q2	27	14	11	30	30	20
			Q3	26	12	35	32	32	18
			Q4	14	20	47	31	31	10
					Produce	Seafood			
					Beverages	Condiments			
					Product (Category)				

Facts

- concept relevant for the analysis: represents a measurement event (typically models a set of events taking place in the company)
- non-empty cell in the cube
- has a granularity = level of detail
- determined by the value of its dimension coordinates

Facts

Transaction fact

- Represent physical world events (at most detailed level)
- Exactly one fact per event
- Typically, events are independent and will occur at any time

Transaction fact example

- One fact for each sale of a wine bottle (detail)
- One fact for each day where at least one bottle sold (aggregation)
- sale fact tied to specific time/place

Snapshot fact

- Measures the current state of a process (possibly the result of a series of events)
- Generally not independent
- Evaluated at specific interval/time

Snapshot fact example

- Inventory level per product,month,store
- a same product may appear in several inventories snapshots

Dimensions

- analysis perspective
- axis of the cube
- described by attributes

Dimensions can be used to define more than one cube.

Dimension attributes form a hierarchy of subsets (containment hierarchy): each level describes one degree of detail for the analysis on this dimension.

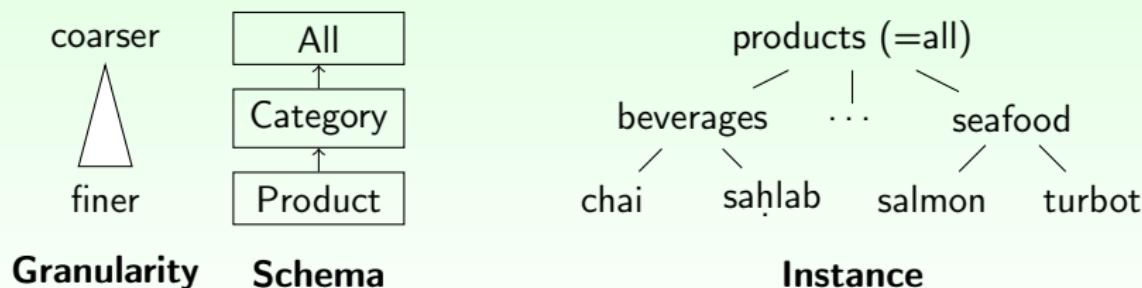
Dimensions

Dimension Schema

The classification schema of a dimension is a partially ordered set of category attributes: $(D.k_1, \dots, D.k_n, \text{Top}_D, \rightarrow)$.

- \rightarrow is the functional dependency
- Top_D is the maximal attribute regarding \rightarrow : $\forall i, D.k_i \rightarrow \text{Top}_D$
- there exists a (unique) minimal attribute: $\exists i, \forall j \neq i, D.k_i \rightarrow D.k_j$.
 $D.k_i$ describes the finest granularity of the dimension.

Classification schema and instance of Product dimension



Dimensions

		Beverages	Produce	Condiments	Seafood
2010	Q1	Jan	4	5	0
		Feb	16	3	28
		Mar	4	10	0
		Apr	5	5	5
	Q2	May	12	5	10
		Jun	13	5	4
		Jul	20	10	7
	Q3	Aug	2	2	6
		Sept	3	3	4
		Oct	8	10	2
	Q4	Nov	3	10	8
		Dec	4	5	8
2011	Q1	Jan	10	25	40
...					

Hierarchy (schema and instance) of temporal dimension?

Dimensions

The definition allows parallel paths in the schema (*parallel hierarchies*)

Classification schema of temporal dimension with (year,month,week,day)?

For this course we assume the classification schema has only one path.
We will also assume that items in distinct dimensions are independant
(rules out $D.k_i \rightarrow D'.k_j$).

Measures

- describes a fact
- consists of two functions:
 - **numeric** property for each fact
 - function to compute measure at coarser aggregation levels

Several Measures can be associated to a fact.

A measure can depend on (other) measures on other facts.

Measure examples:

- number of units in stock
- value per unit
- quantity \times price \times turnover
- ...

Cube

Cube

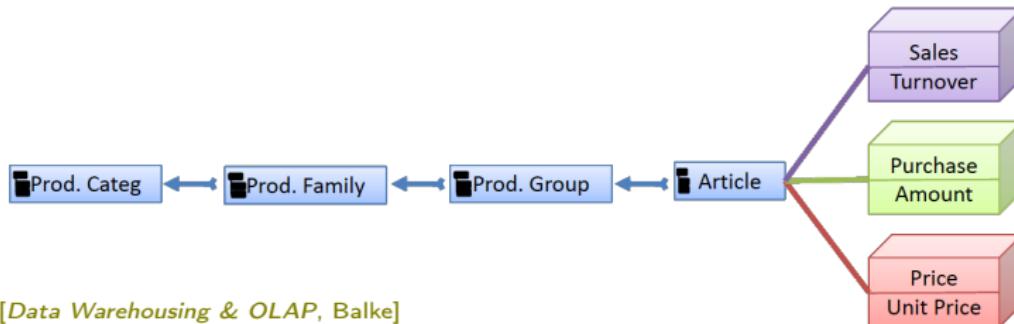
Cube Schema: set of n dimension schemas, and m measures $(\{D_1, \dots, D_n\}, \{M_1, \dots, M_m\})$.

Cube: instance of the cube schema = set of cells in $\text{dom}(D_1) \times \dots \times \text{dom}(D_n) \times \text{dom}(M_1) \times \dots \times \text{dom}(M_m)$

The cube contains *all* cells from its domain, not only the non-empty ones.

Can be observed at several granularities (determined by one level on each dimension)

Reminder: several cubes may share a dimension.



Outline

M2 D&K 2017

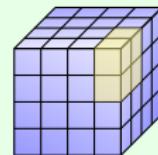
③ DW concepts: the multi-dimensional model

- Components overview
- Multidimensional data model: facts, measures, dimensions, cube
- Multi-dimensional query concepts: OLAP operations

OLTP vs OLAP: queries

Queries

Operational DB	DW
transactions	analytical queries
INSERT, UPDATE, SELECT	SELECT, bulk insert
query deals with few tuples (often 1)	drill-down, slice...)



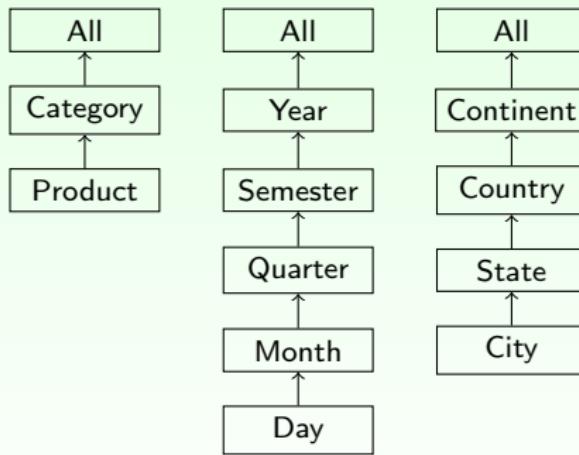
```
UPDATE Customers  
SET City='Heidelberg'  
WHERE CustomerName='H. Plattner';
```

Multidimensional model

We next illustrate operations on:

Cube, and dimension schemas

		Köln				Berlin				Lyon				Paris						
		24	18	28	14	33	25	23	25	12	20	24	33	25	18	21	10	18	35	
Customer (City)	Time (Quarter)	Q1	21	10	18	35	35	33	23	25	14	12	20	33	25	18	21	10	18	35
		Q2	27	14	11	30	30	32	20	17	12	18	25	33	20	18	27	14	11	30
		Q3	26	12	35	32	32	10	33	18	10	33	23	31	20	18	26	12	35	32
		Q4	14	20	47	31	31	31	31	31	31	31	31	31	31	31	14	20	47	31
			Produce	Seafood	Beverages	Condiments	Product (Category)													



Restrictions:

- we consider a single cube
- this cube has only 3 dimensions
- we consider a single measure (sales quantity)

OLAP operations

Typical OLAP operations (cube navigation):

- Roll-up
- Drill-down
- Slice and Dice

Typical OLAP operations (rearranging the cube):

- Pivot
- Sort

Advanced OLAP operations:

- Drill-across
- Drill-through

Other common operations: aggregate functions, ranking functions. . .

Roll-up

In short; *roll-up* summarizes data by navigating upward in the hierarchy.

Ex: (category,city,day) → (category,country,year).

Roll-up/Drill-up/Consolidate

Let C a cube with schema $(\{D_1, \dots, D_n\}, \{M_1, \dots, M_m\})$ at granularity $G = (l_1, \dots, l_n)$, where $1 \leq l_i < m + 1$ is the level in dimension $D_i : (D_i.k_1, \dots, D_i.k_m, \text{Top}_{D_i}, \rightarrow)$.

A Roll-up operation navigates toward a coarser granularity: some dimension $\text{Dim}^{up} \subset \{D_1, \dots, D_n\}$ are rolled-up;

The new granularity is $G' = (l'_1, \dots, l'_n)$.

- $\forall D_i \in \text{Dim}^{up}, l_i < l'_i \leq m + 1$
- $\forall D_i \notin \text{Dim}^{up}, l_i = l'_i$

This definition allows

- to zoom-out on multiple dimensions at once
- to roll-up a dimension to the Top_D level*

*sometimes called *Drill-in* or *Merge*

Roll-up

ROLLUP(Cube, Dimension → Level, AggFunction(Measure))

The diagram illustrates the roll-up process from a detailed 4x4 cube to a summary 2x2 cube. A yellow arrow points from the detailed cube on the left to the summary cube on the right.

Detailed Cube (Left):

Customer (City)	Time (Quarter)	Product (Category)		Köln				Berlin				Lyon				Paris				
		Beverages	Condiments	24	18	28	14	33	25	23	25	14	12	20	24	33	21	10	18	35
Time (Quarter)	Q1	21	10	18	35	35	23	25	14	23	18	30	32	12	20	17	21	10	18	35
	Q2	27	14	11	30	30	14	23	20	12	18	32	32	10	33	18	21	10	18	35
	Q3	26	12	35	32	32	12	23	20	10	33	31	31	10	33	18	21	10	18	35
	Q4	14	20	47	31	31	21	23	20	17	18	31	31	17	31	18	21	10	18	35

Summary Cube (Right):

Customer (Country)	Time (Quarter)	Product (Category)		Germany				France											
		Beverages	Condiments	57	43	51	39	33	30	42	68	68	41						
Time (Quarter)	Q1	33	30	42	68	33	30	42	68	68	41	44	37	44	51	41	41	41	41
	Q2	39	26	41	44	39	26	41	44	44	44	41	37	44	51	41	41	41	41
	Q3	30	22	46	44	30	22	46	44	44	44	41	51	44	51	41	41	41	41
	Q4	25	29	49	41	25	29	49	41	41	41	41	41	41	41	41	41	41	41

Roll-up to the Country level on Customer dimension:

ROLLUP(Sales, Customer → Country, SUM(Quantity))

Remark: beware semi-additive or non-additive measures.

Roll-up

Draw the result of the operation.

Köln	24	18	28	14			
Berlin	33	25	23	25			
Lyon	12	20	24	33			
Paris	10	18	35				
Q4		10	18	35	35	33	18
Q3	27	14	11	30	30	14	23
Q2	26	12	35	32	32	12	20
Q1	14	20	47	31	31	10	33
	Produces		Seafood				
Beverages							
			Condiments				

ROLLUP(Sales, Time→ Semester, SUM(Quantity))

Roll-up

Draw the result of the operation.

Köln	24	18	28	14	
Berlin	33	25	23	25	14
Lyon	12	20	24	33	
Paris	10	18	35	25	18
Q4		10	18	35	35
Q3	27	14	11	30	30
Q2	26	12	35	32	32
Q1	14	20	47	31	31
	Produces	Seafood			
Beverages					
Condiments					

ROLLUP(Sales, Customer → Country, Time → Top_{Time},
SUM(Quantity))

Hierarchical rollup, Dimensional rollup*, or both simultaneously.

* sometimes called Drill-out or Split

Drill-down

Inverse operation of Roll-up: zoom-in on the data.

Ex: (category,country,year) \rightarrow (category,city,day).

Drill-Down

Let C a cube with schema $(\{D_1, \dots, D_n\}, \{M_1, \dots, M_m\})$ at granularity $G = (l_1, \dots, l_n)$, where $1 \leq l_i < m + 1$ is the level in dimension $D_i : (D_i.k_1, \dots, D_i.k_m, \text{Top}_{D_i}, \rightarrow)$.

A Drill-down operation navigates toward a finer granularities: some dimension $Dim^{down} \subset \{D_1, \dots, D_n\}$ are drilled-down;

The new granularity is $G' = (l'_1, \dots, l'_n)$.

- $\forall D_i \in Dim^{down}, 1 \leq l'_i < l_i$
- $\forall D_i \notin Dim^{down}, l_i = l'_i$

It's obvious but... you cannot drill down if you do not have the finer-grained data!

Drill-down

DRILLDOWN(Cube, Dimension→ Level)

Customer (City)	Köln	24	18	28	14	14
		33	25	23	25	
Berlin	12	20	24	33	25	18
Lyon	12	20	24	33	25	18
Paris	21	10	18	35	33	18
Q1	21	10	18	35	35	17
Q2	27	14	11	30	30	12
Q3	26	12	35	32	32	18
Q4	14	20	47	31	31	10
	Produce	Seafood				
	Beverages	Condiments				
Product (Category)						



Customer (City)	Köln	8	6	9	5	5
	Berlin	10	8	11	8	
Lyon	4	7	8	14	8	6
Paris	7	2	6	20	20	10
Jan	7	2	6	20	20	10
Feb	8	4	8	8	8	7
Mar	6	4	4	7	7	...
...
Dec	4	4	16	7	7	14
	Produce	Seafood				
	Beverages	Condiments				
Product (Category)						

Drill-down to the Month level on Time dimension:

DRILLDOWN(Sales, Time→ Month)

Drill-down/Roll-up Summary

Navigate between granularities.

- Roll-up: fewer details
measure may be computed from input cube
- Drill-down: more details
measure computed from the finest-detailed data

... assuming measures have not (yet) been materialized at other granularities.

The number of dimensions remains the same (except when at top level of dimension).

Slice and Dice

Slice and Dice definitions

Slice: returns a “slice” of the cube by selecting a *single* value on *one* of the dimensions.

⇒ corresponds to SQL’s WHERE with equality selection.

Dice: returns a “dice” of the cube by selecting for each dimension a boolean combination of range or value conditions on one dimension *single* value.

 definitions vary between authors.

Slice

SLICE(Cube,Dimension, Level=value)

The diagram illustrates the concept of slicing a multidimensional cube. On the left, a 4D cube is shown with dimensions: Customer (City) [Köln, Berlin, Lyon, Paris], Time (Quarter) [Q1, Q2, Q3, Q4], Product (Category) [Beverages, Condiments, Produce, Seafood], and Sales Value. The values are represented by a 4x4 grid of numbers. A yellow arrow points from this cube to a smaller 3D cube on the right, which represents the result of slicing the original cube along the Customer dimension where City='Paris'. This resulting cube has dimensions Time (Quarter), Product (Category), and Sales Value.

	Köln	24	18	28	14
	Berlin	33	25	23	25
	Lyon	12	20	24	33
	Paris	21	10	18	35
Time (Quarter)	Q1	21	10	18	35
	Q2	27	14	11	30
	Q3	26	12	35	32
	Q4	14	20	47	31
Product (Category)		Produce	Seafood		
Beverages					
Condiments					

	Q1	21	10	18	35
	Q2	27	14	11	30
	Q3	26	12	35	32
	Q4	14	20	47	31
Product (Category)		Produce	Seafood		
Beverages					
Condiments					

Slice on City='Paris':

SLICE(Sales, Customer, City='Paris')

Dice

DICE(Cube, Φ), with Φ : boolean combination

The diagram illustrates a DICE operation on a 4D cube. The original cube has dimensions: Customer (City) [Köln, Berlin, Lyon, Paris], Time (Quarter) [Q1, Q2, Q3, Q4], Product (Category) [Beverages, Condiments, Produce, Seafood], and Sales Value. The reduced cube on the right retains dimensions Customer (City) [Lyon, Paris] and Time (Quarter) [Q1, Q2], and includes the columns for Produce and Seafood from the original cube.

		Customer (City)		Time (Quarter)					
		Köln	Berlin	Q1	Q2	Q3	Q4	Product (Category)	Sales
Customer (City)	Köln	24	18	28	14			Beverages	33
	Berlin	33	25	23	25				
Time (Quarter)	Lyon	12	20	24	33	35	33	Condiments	21
	Paris	21	10	18	35	35	25		
		21	10	18	35	35	23	Produce	18
		27	14	11	30	30	20	Seafood	17
		26	12	35	32	32	18		
		14	20	47	31	31	33		

Dice on City='Paris' or 'Lyon' and Quarter='Q1' or 'Q2':

DICE(Sales,(Customer.City='Paris' OR Customer.City='Lyon')
AND (Time.Quarter='Q1' OR Time.Quarter='Q2'))

Other operations

A few more typical operations on data:

- sorting the cube
- pivoting
- joining cubes

→ but no official standard for OLAP queries.

Cross-tabulations

Also called crosstab, pivot table, contingency table.

Simple tool to help find interactions between 2 variables.

Sales per city and day							
	Mon	Tue	Wed	Thu	Fri	Sat	Row totals
Paris	10	20	30	40	30	10	140
Lyon	40	20	20	30	50	30	190
Lille	50	20	30	20	10	0	130
Col totals	100	60	80	90	90	40	460

We can view pivoting as selecting 2 dimensions to aggregate some measure.
Can be computed in Excel, and similar operation in many SQL implementations.

OLAP visualization in practice

Screenshot of SAP Business Object interface showing OLAP visualization.

The interface includes:

- Top navigation bar: Home, Documents, IDES BW Sales Anal... (with a dropdown arrow), and several icons.
- Toolbar: Filter, Sort, Calculations, Conditional Formatting, and Auto Update.
- Left sidebar: Data (with a tree view of "IDES BW Sales Analysis"), Key Figures (Orders Sold, Net Value, Open Orders, Cal. year / month, Calendar Year, Region, Division, Sales organization, Sold-to party, Sales Group, Product, Country, Distribution channel), and Background (2008 (YeaQuaMon) and Orders Sold (Key Figu)).
- Middle section: Layout panel with Columns (Distribution channel, Division) and Rows (PM Sales Organization). The main area displays "Analysis 1" with the following data:

PM Sales Organization	Distribution channel	Division
Final Customer Sales 15 : Electronic Parts 7 : High Tech		
Europe	8,964	12,790
Germany	2,940	1,260
France	1,676	3,572
GB	3,700	6,470
Switzerland	648	1,488
Americas	4,688	16,000
USA	4,064	15,262
Canada	624	738

Below the analysis, there is a toolbar with "Copy (5) of Analysis 1" and a "Delete Analysis" button.

At the bottom, two pie charts are displayed:

- 2008: 37.17% (purple), 23.16% (blue), 16.64% (green), 16.64% (orange).
- Q1 2008: 32% (purple), 29.45% (blue), 16.64% (green), 16.64% (orange).

Legend for the charts:

- Germany (Blue)
- France (Orange)
- GB (Green)
- Switzerland (Red)
- USA (Purple)
- Canada (Yellow)

Bottom navigation: Sheet 1, Sheet 2, Sheet 3.

SAP Business Object [<https://www.youtube.com/watch?v=3rhiRjH6LiA>]

OLAP visualization in practice(2)

New Workspace - IBM Cognos Insight

Style Insert Get Data Explore

Channel profitability Measures Total of Month Central Europe Lanterns

	Actual			Target			Count					
	Total of Total of Month		2012	2013	Total of Total of Month		2012	2013	Total of Total of Month		2012	2013
	Golf Shop	105,928	55,968	49,960	102,311	48,536	53,775	112	56	56	56	
Department Store	13,511	6,856	6,655	10,864	5,196	5,668	22	11	11	11		
Direct Marketing	8,317	4,407	3,910	7,941	4,172	3,769	16	8	8	8		
Warehouse Store	17,278	9,161	8,117	15,339	7,788	7,551	24	12	12	12		
Equipment Rental Store	270	141	129	226	132	93	2	1	1	1		
Outdoors Shop	55,936	30,028	25,908	56,040	24,746	31,294	24	12	12	12		
Eyewear Store												
Sports Store	10,816	6,375	5,241	11,902	6,502	5,400	24	12	12	12		

Actual Target Count

The figure consists of three separate bar charts sharing a common vertical axis. The left chart shows Actual values, the middle shows Target values, and the right shows Count values. Each chart has two groups of bars for the years 2012 and 2013. Within each group, there are seven bars representing different store categories: Golf Shop (blue), Department Store (orange), Direct Marketing (green), Warehouse Store (light green), Equipment Rental Store (dark blue), Outdoors Shop (light blue), Eyewear Store (purple), and Sports Store (pink). The Y-axis for all three charts ranges from 0 to 35,000.

Golf Shop Department Store DirectMarketing Warehouse Store Equipment Rental Store Outdoors Shop Eyewear Store Sports Store

Toolbox

IBM Cognos [<https://www.youtube.com/watch?v=FjKaRU5V1Rw>]

OLAP visualization in practice(3)

Book1 - Microsoft Excel

PivotTable Tools Options Design

PivotTable Name: Active Field: Year

PivotTable1

Options PivotTable

Field Settings Active Field

Group Selection Group

Group Field Group

Sort Sort & Filter

Refresh Change Data Source

Clear Actions

Select PivotTable

Summarize Values By

Show Values As Calculations

Fields, Items, & Sets

PivotChart OLAP Tools What-If Analysis

Field List Buttons Field Headers

C2 2011

	A	B	C	D	E	F	G	H	I	J	K	L
1	Net Sales	Column Labels	2011									
2	Row Labels	2010	2011	Grand Total								
3	JR - John Roberts		55,127.97	55,127.97								
4	JR - Juan Roca		330,285.42	330,285.42								
5	MS - Miguel Severino	22,140.00	413,059.65	435,199.65								
6	PS - Peter Saddow	3,695.70	68,523.19	72,218.89								
7	Grand Total	25,835.70	866,996.23	892,831.93								
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												
21												
22												
23												
24												
25												

PivotTable Field List

Show fields related to: (All)

- Global Dimension 2
 - Dimension 2
- Global Dimension 3
 - Dimension 3
- Item
 - Item by Category by Product Group
 - Item by Gen Prod Posting Group
 - Item by Inventory Posting Group

Drag fields between areas below:

Report Filter Column Labels Date YQMD

Row Labels Salesperson-Purchaser

Values Net Sales

Defer Layout Update Update

Sheet1 Sheet2 Sheet3

Ready

5:27 PM 9/30/2012

Excel [<https://www.youtube.com/watch?v=ygAs-6mEhBA>]

Outline

M2 D&K 2017

④ Relational implementation: schemas

- Schemas
- Relational implementation: dimensional modeling
- Fact table
- Dimension hierarchies
- Dimension attributes
- Dimension variations
- Dimension implementation in Oracle

Outline

M2 D&K 2017

4 Relational implementation: schemas

- Schemas
 - Alternative approaches to store the cube
 - Star and snowflake schemas
 - Additional schema variants
- Relational implementation: dimensional modeling
- Fact table
- Dimension hierarchies
- Dimension attributes
- Dimension variations
- Dimension implementation in Oracle

Schema

Storage

Operational DB	DW
3NF schema, no redundancy	redundant information
schema independant of query	multidimensional model: dimensions and measures based on end-user queries
detailed data	pre-aggregated data
MB,GB	TB,PB

Implementing the cube

3 main approaches to implement the multidimensional model

- **ROLAP**: stores the cube in relational databases. Extensions of SQL to implement model and OLAP operations
- **MOLAP**: stores the cube in specific datastructure (array)
- **HOLAP**: combines both storage technologies

Implementing the cube

3 main approaches to implement the multidimensional model

- **ROLAP**: stores the cube in relational databases. Extensions of SQL to implement model and OLAP operations
- **MOLAP**: stores the cube in specific datastructure (array)
- **HOLAP**: combines both storage technologies

ROLAP

Pro:

- stores only non-empty cells
- dimension tables typically fit in memory
- benefits from relational technology

Con:

- cost of translating into SQL

MOLAP

Pro:

- need not store cell coordinates
- accessing particular cell easy (array position)
- good performance for queries and aggregation (including aggregation maintenance)
- some analytical queries are not supported in SQL
- allows to specify access control options depending on the granularity

Con:

- lack of standard, limited offer (proprietary)
- storing aggregates generates overhead
- limited storage capacity w.r.t. mature relational technology
- scales poorly with dimensions

ROLAP

Aim: implement the cube; dimensions, measures... in relational system.

- keep semantic information (dimension hierarchies)
- keep maintenance simple
- support efficiently multidimensional queries (translation into SQL, execution...)
- traditional DBMS requirements: performance, concurrent access, security...

ROLAP: characteristics

- stores data in relations
- separation between structure and content:
 - one relation stores all facts
 - other relations store dimension attributes
 - the fact relation stores measures and references the dimension tables
- only stores existing facts

Typically, schema adopts traditional patterns:

- star schema
- snowflake schema
- (galaxy schema)
- starflake schema

Star schema

Star schema

A **star schema** is defined by a *fact table* and a set of *dimension tables*

- Each dimension $(D_i.k_1, \dots, D_i.k_l, \text{Top}_D, \rightarrow)$ is stored in a relation D_i of schema $(\underline{PK}, k_1, \dots, k_l)$
- The fact relation has schema
 $\underline{(FK_1 \rightarrow D_1.PK, \dots, FK_n \rightarrow D_n.PK, M_1, \dots, M_m)}$

PK is key of D_i . One attribute per level in the schema.

The combination of all FK is key of the fact relation. One attribute per measure.

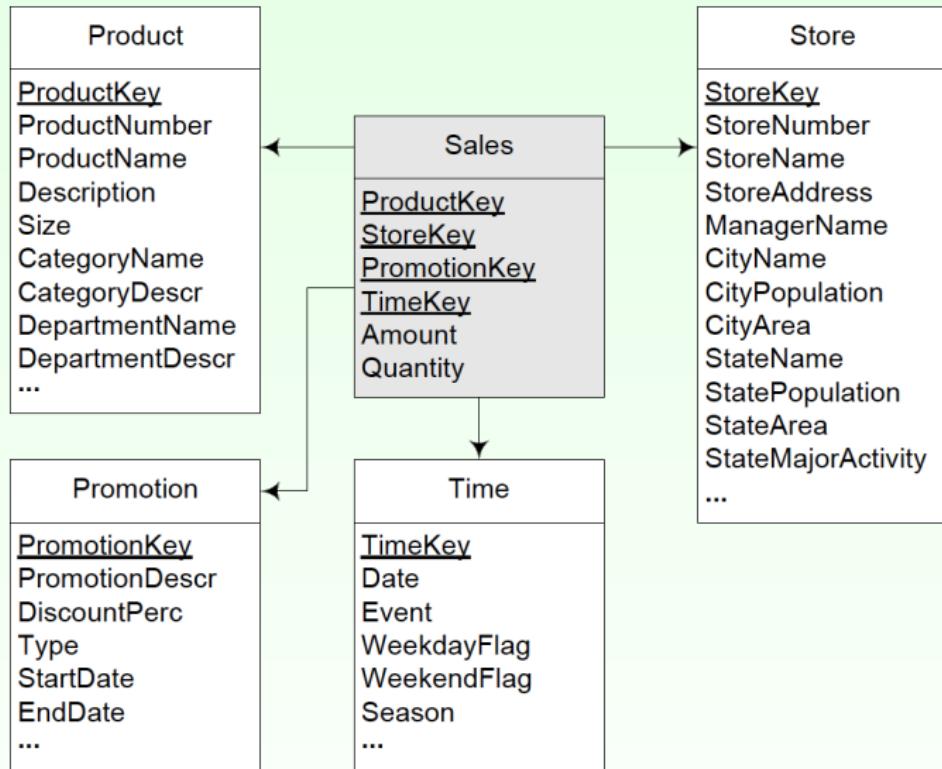
Star schema: properties

- dimension relations typically in 2NF: redundancy
- dimension relations contain relatively few tuples compared to fact: redundancy is not expensive
- fact relation in 3NF
- typically use surrogate keys for dimension key
- efficient queries

Star schema



Star schema example



Snowflake schema

Snowflake schema

A **snowflake schema** is defined by a *fact table* and a set of *dimension tables*

- Each dimension $(D_i.k_1, \dots, D_i.k_l, \text{Top}_D, \rightarrow)$ is stored in l relations D_i^1, \dots, D_i^l , where relation D_i^j has schema $(\underline{PK}, A_1, A_2, \dots, FK \rightarrow D_i^{j+1})^\dagger$
- The fact relation has schema $(FK_1 \rightarrow D_1.PK, \dots, FK_n \rightarrow D_n.PK, M_1, \dots, M_m)$

PK is key of D_i . One attribute per level in the schema.

The combination of all FK is key of the fact relation. One attribute per measure.

There may be several FK in dimension tables.

[†]of course PK, FK and the attributes $A_1, A_2 \dots$ are generally distinct between 2 relations

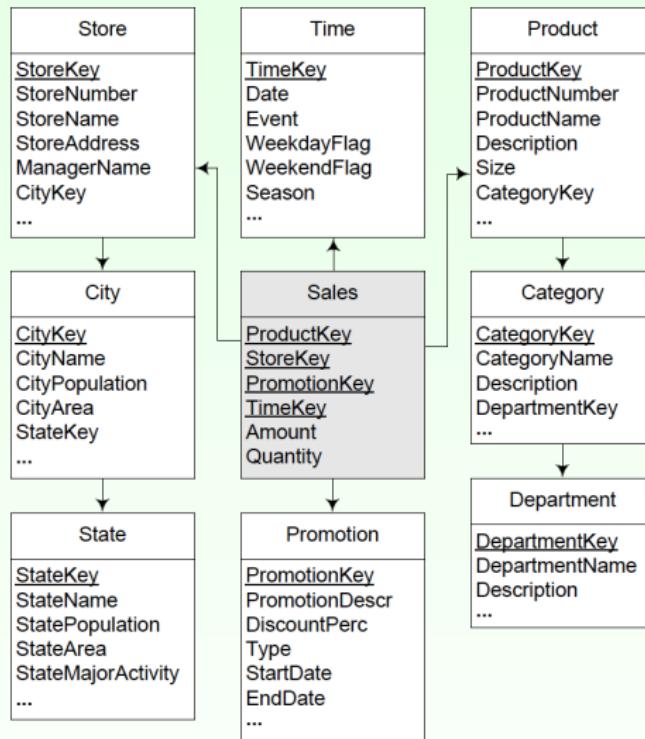
Snowflake schema: properties

- redundancy avoided with normalized dimension tables
- smaller dimension tables (might speedup some queries)
- n hierarchy levels in dimensions $\implies n$ dimension tables
- fact relation in 3NF
- fact table references dimension table with finest granularity
- queries perform many joins: performance degraded

Snowflake schema



Snowflake schema example



Snowflakes

Pro:

- Normalization reduces the size of dimension table (may improve query time)
- Normalization facilitates maintenance
- better represent hierarchy

Con:

- Space gained by normalization generally negligible
- Normalizations makes query processing harder (joins)

⇒ *Base decision on granularity of typical queries, size of dimension table, frequency of updates, materialization.*

Outrigger dimension

... not a schema type!



Outrigger

A table that is joined to other dimension tables in a star schema.

An outrigger can be used to snowflake a dimension, but unlike snowflakes needs not be 3NF, and is only a 2nd layer.

Typical usage: table that is shared by more than one dimension.

Galaxy schema

Galaxy schema

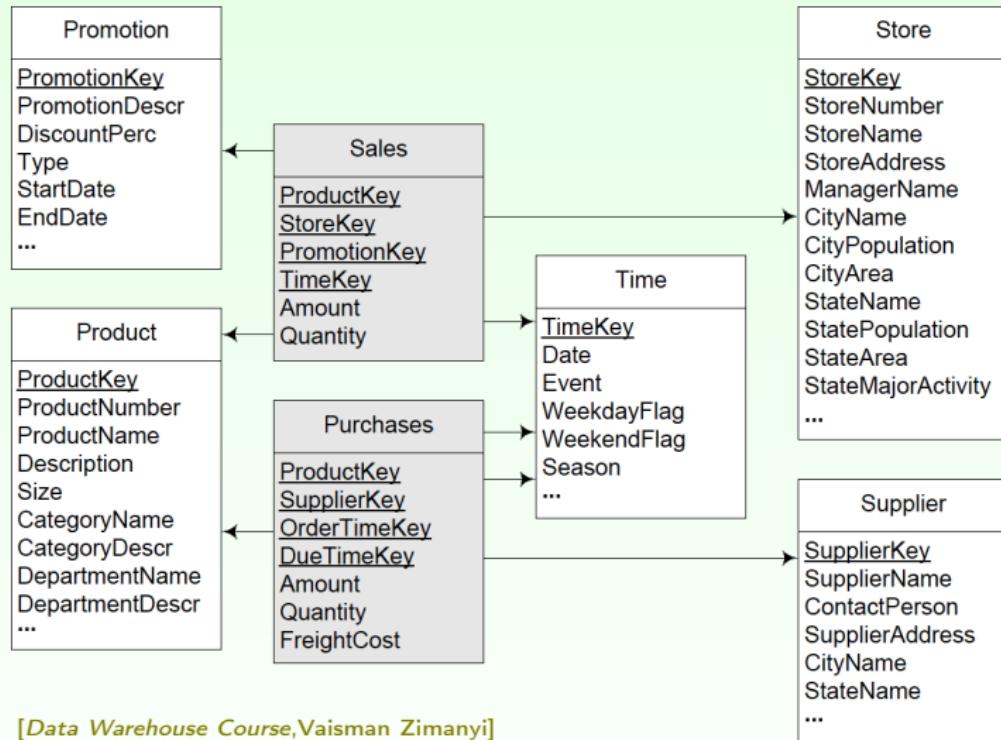
A **galaxy schema** is a collection of star schema where several fact table coexist and share some dimensions.

aim: model warehouses where measures are not all described by the same dimensions

 Some authors call galaxy schemas “constellation schemas”.

Galaxy schema

Galaxy schema example



Galaxy schema: fact constellation

Particular use case for galaxy schema: storing aggregate values to optimize performance.

Fact constellation example



Improves performance and aggregate (possibly also detailed) data.

Starflake schema

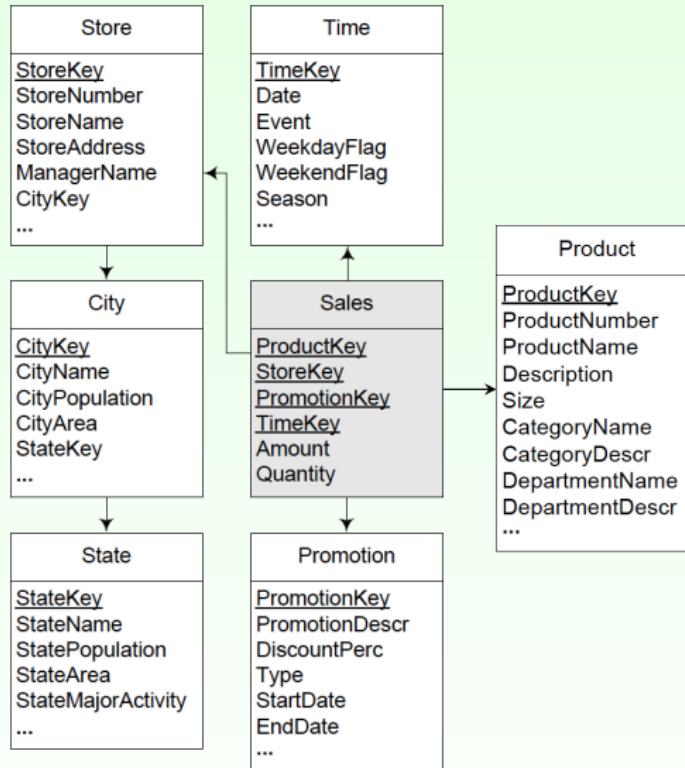
Starflake schema

A **starflake schema** is a combination between a star schema and a snowflake schema: some dimensions are normalized, but not all.

aim: leverage benefits of both star and snowflake schemas

Starflake Schema

Starflake schema example



Outline

M2 D&K 2017

④ Relational implementation: schemas

- Schemas
- Relational implementation: dimensional modeling
- Fact table
- Dimension hierarchies
- Dimension attributes
- Dimension variations
- Dimension implementation in Oracle

Outline

M2 D&K 2017

④ Relational implementation: schemas

- Schemas
- Relational implementation: dimensional modeling
- Fact table
- Dimension hierarchies
- Dimension attributes
- Dimension variations
- Dimension implementation in Oracle

Nulls

[Kimball]

- NULL ok for *measures*
- NULL must be avoided in Fact table's *foreign keys*.

Why?

- introduce a default row in dimension tables
- NULL in dimension attributes: better replace with explanation string

Fact table taxonomy

[Kimball]

- transaction fact tables
- periodic snapshot fact table
- accumulating snapshot fact table
- factless fact table
- aggregate fact table
- consolidated fact table

Outline

M2 D&K 2017

④ Relational implementation: schemas

- Schemas
- Relational implementation: dimensional modeling
- Fact table
- Dimension hierarchies
- Dimension attributes
- Dimension variations
- Dimension implementation in Oracle

Hierarchies are not always simple...

Hierarchies may

- be unbalanced (some instance members have no child)
- be ragged (some paths may skip some levels) ex : some city have no state
- be recursive ex : employee.managerkey references (another) employee.id
- admit parallel or mutually exclusive paths

Typical approaches: introduce placeholders/nulls for missing levels, record transitive closure, add n-n bridge tables, split the dimension.

Recursive hierarchy queries are now supported in multiple tools, esp. RDBMS (CTE, CONNECT BY).

Outline

M2 D&K 2017

④ Relational implementation: schemas

- Schemas
- Relational implementation: dimensional modeling
- Fact table
- Dimension hierarchies
- **Dimension attributes**
- Dimension variations
- Dimension implementation in Oracle

Choosing attributes

Dimension attributes = possible constraints in the WHERE and GROUP BY clauses of the SQL queries.

⇒ choosing right attributes crucial to offer analysis capabilities!

Attribute vs measure?

Rule of thumb:

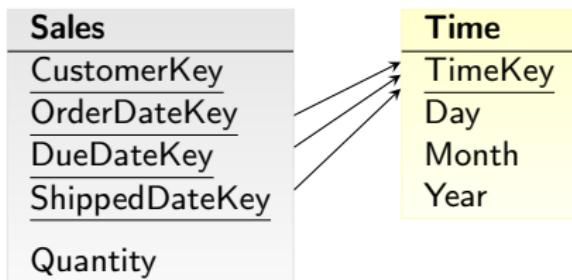
- continuous → measure
- small list → dimension.

Role-playing dimensions

Role-playing dimension

a dimension that participates several times in a fact table

- some OLAP product may forbid roles: need to create one dimension per role.
- SQL aliases (often supported by BI tools)
- keep **single dimension** but create **one view per role**. Relabel attributes to distinguish between views.

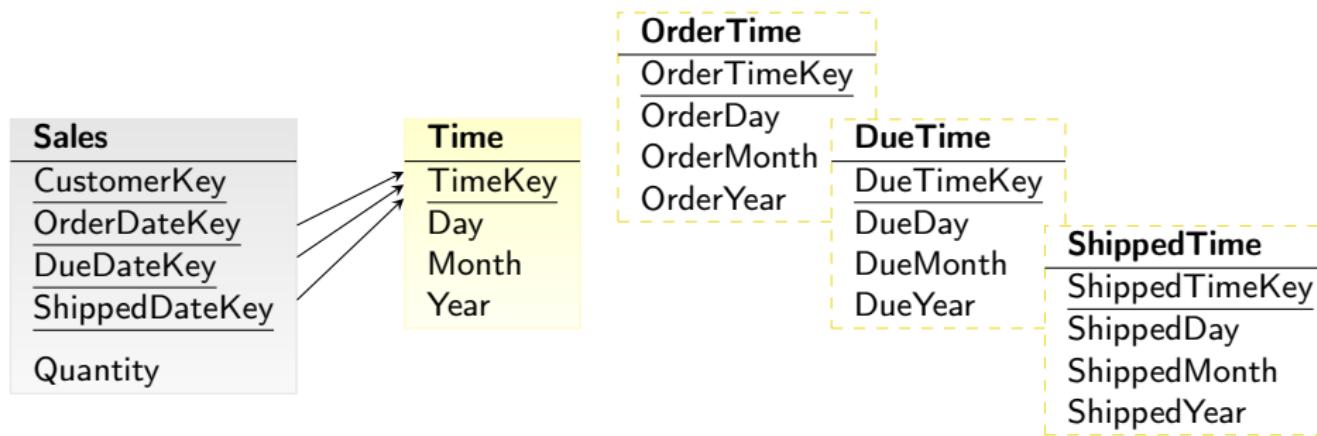


Role-playing dimensions

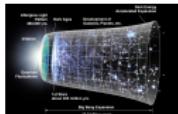
Role-playing dimension

a dimension that participates several times in a fact table

- some OLAP product may forbid roles: need to create one dimension per role.
- SQL aliases (often supported by BI tools)
- keep **single dimension** but create **one view per role**. Relabel attributes to distinguish between views.



Time dimension



- Standard practice: use a date dimension:
 1. some information not supported by the SQL date functions (holidays, fiscal periods, working days)
 2. easier to store than compute on the fly, esp. for business users unfamiliar with SQL
- in general *time of day* is a measure of the fact table
- if need particular rollups on time of day, make it a separate dimension

Flags/indicator attributes

Keep textual indicators explicit in the dimension tables (avoid 'Y/N').

Ex:

Date

DateKey	Date	Day of Week	Calendar Month	Quarter	Calendar Year	Holiday Indicator	Weekday Indicator
20150101	01/01/2015	Thursday	January	Q1	2015	Holiday	Weekday
20150102	02/01/2015	Friday	February	Q1	2015	non-Holiday	Weekday
:	:	:	:	:	:	:	:

If many low cardinality indicators, store them all in a single *junk dimension*.

Keys

Primary Key of dimensions should be *surrogate keys*

- independance from data source/integration
- performance
- allow to track changes in dimensions



Outline

M2 D&K 2017

④ Relational implementation: schemas

- Schemas
- Relational implementation: dimensional modeling
- Fact table
- Dimension hierarchies
- Dimension attributes
- **Dimension variations**
- Dimension implementation in Oracle

Slowly changing dimension

In practice dimensions are modified (instance and schema).

- there was an error \implies should replace old data (instance level)
- context has changed

But changes are occasional.

Slowly changing dimension

a dimension designed to support occasional changes.

 of course insertion of new *facts* is not the issue here.

Multiple approaches. First one is attribute of type 0:

- forbids update to attribute values
- allows insertion of new values
- used (seldom) for persistent attribute; initial credit amount, durable key, creation date.

SCD : illustrative example

Sale

Date Key	Customer Key	Product Key	Sales Amount
t1	c1	1	100
t2	c2	1	100
t3	c3	3	100
t4	c4	4	100

Dimension changes:

- new products
- modify product info (error)
- update category (evolution)

Product

Product Key	Product Name	Category
1	Cinnamon Roll	Bread
2	Baked Whole Wheat Bread	Bread
3	Treacle Tart 320 g	Desserts
4	Pecan Butter 64 fl oz	Desserts

After t4, the category of *Cinnamon Roll* is set to *Dessert*.

SCD (1)

Type 1 attribute: overwrite

- allows to update the value of attribute (old row is overwritten).
- used to correct erroneous values.

Product

Product Key	Product Name	Category
1	Cinnamon Roll	Dessert
2	Baked Whole Wheat Bread	Bread
3	Treacle Tart 320 g	Desserts
4	Pecan Butter 64 fl oz	Desserts

SCD (2)

Type 2 dimension: versioning (with row)

- keep track of historical records by version validity interval (or version number)
- when attribute value changes, insert new row with same natural key (different surrogate)
- unlimited number of historical records
- fact table untouched (old facts reference old row)

Product

Product Key	Product Name	Category	Row Effective Date	Row Expiration Date	Current Row Indicator
1	Cinnamon Roll	Bread	2010-01-01	2015-08-20	Expired
5	Cinnamon Roll	Dessert	2015-08-21	9999-12-31	Current
2	Baked Whole Wheat Bread	Bread	2010-01-01	9999-12-31	Current
3	Treacle Tart 320 g	Desserts	2010-01-01	9999-12-31	Current
4	Pecan Butter 64 fl oz	Desserts	2010-01-01	9999-12-31	Current

SCD (3)

Type 3 dimension: versioning (with attribute)

- parallels current and old with additional column(s)
- overwrite attribute value, save old value in “prior value” column
- limited history

Product

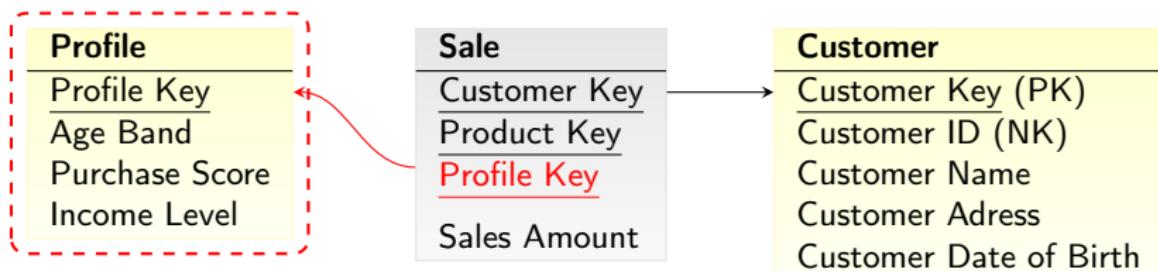
Product Key	Product Name	Category	Prior Category
1	Cinnamon Roll	Dessert	Bread
2	Baked Whole Wheat Bread	Bread	NULL
3	Treacle Tart 320 g	Desserts	NULL
4	Pecan Butter 64 fl oz	Desserts	NULL

SCD (4)

Type 4 dimension: mini-dimension

- store most frequently updated (or queried) attributes in a mini-dimension
- one row per combination of values appearing in the data = “profiles”
- used to deal with very large dimensions (a.k.a. *monster dimensions*)
- when attribute update: insert new profile (if needed), subsequent facts will use the new profile.

Mini-dimension



⚠ Other type 4 definition in literature: keep current data in the dimension, store separate *history table* as in type 2.

Outline

M2 D&K 2017

④ Relational implementation: schemas

- Schemas
- Relational implementation: dimensional modeling
- Fact table
- Dimension hierarchies
- Dimension attributes
- Dimension variations
- Dimension implementation in Oracle

Dimensions in Oracle

[Oracle Database Online Documentation 12c (Data Warehousing)]

Dimension definition (Oracle)

```
CREATE DIMENSION name_dimension
  <level definition clauses>
  <hierarchy definition clauses>
  <attribute dependencies clauses>
```

Dimensions in Oracle SQL: star

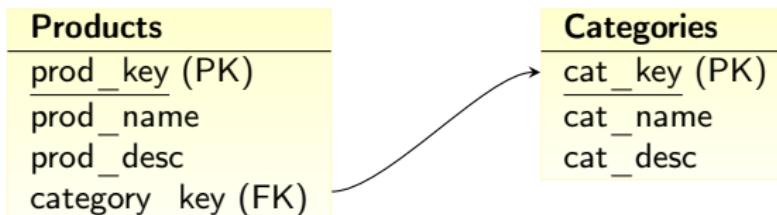
Time

date
day_of_week
holiday_flag
calendar_month
calendar_month_number_in_year
calendar_year
fiscal_month
fiscal_year

Dimension definition

```
CREATE DIMENSION date_dim
  LEVEL day IS time.date
  LEVEL month IS time.calendar_month
  LEVEL year IS time.calendar_year
  LEVEL fis_month IS time.fiscal_month
  LEVEL fis_year IS time.fiscal_year
  HIERARCHY cal_rollup (
    day CHILD OF
    month CHILD OF
    year
  )
  HIERARCHY fis_rollup (
    day CHILD OF
    fis_month CHILD OF
    fis_quarter
  )
  ATTRIBUTE day DETERMINES
    (day_of_week, holiday_flag)
  ATTRIBUTE month DETERMINES
    (calendar_month_number_in_year)
;
```

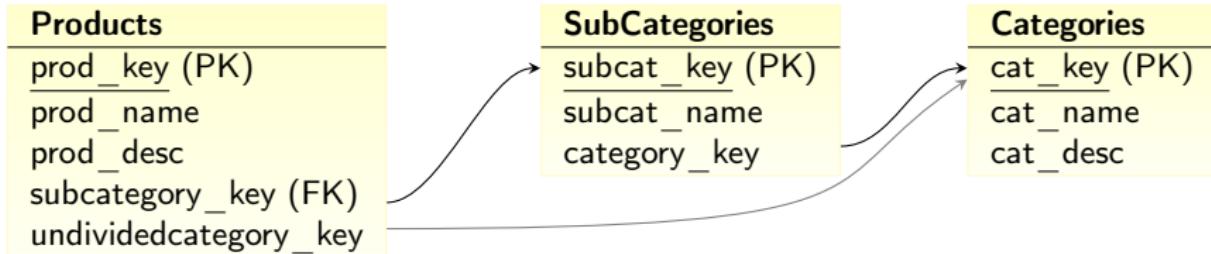
Dimensions in Oracle SQL: snowflake



Dimension definition

```
CREATE DIMENSION products_dim
  LEVEL product IS products.prod_key
  LEVEL category IS categories.cat_key
  HIERARCHY prod_rollup (
    product CHILD OF
    category
  JOIN KEY (product.category_key) REFERENCES category
)
ATTRIBUTE product DETERMINES
  (prod_name, prod_desc)
ATTRIBUTE category DETERMINES
  (cat_name, cat_desc)
;
```

Dimensions in Oracle SQL: snowflake (2)



Dimension definition (Oracle)

```
CREATE DIMENSION products_dim
  LEVEL product IS products.prod_key
  LEVEL subcategory IS subcategories.subcat_key SKIP WHEN NULL
  LEVEL category IS categories.cat_key
  HIERARCHY prod_rollup (
    product CHILD OF
    category
    JOIN KEY (products.subcategory_key) REFERENCES subcategory
    JOIN KEY (subcategories.category_key) REFERENCES category
    JOIN KEY (product.category_key) REFERENCES category
  )
;
```

Dimensions in Oracle SQL

Modifying dimensions (Oracle)

```
ALTER DIMENSION products_dim DROP HIERARCHY prod_rollup;
ALTER DIMENSION products_dim ADD LEVEL brand IS categories.brand;
ALTER DIMENSION products_dim COMPILE; -- to re-validate
DROP DIMENSION products_dim;
...
```

Dimensions in Oracle SQL: usage

Dimensions are viewed in Oracle as constraints, used for query rewriting.

Validating dimensions

```
BEGIN  
    DBMS_DIMENSION.VALIDATE_DIMENSION(  
        dimension => 'products_dim',  
        incremental => false,  
        check_nulls => true,  
        statement_id=>'validation run 1');  
END;  
/
```

- incremental=true: validate new rows only
- check_nulls=true: NULLS occur only in columns with level 'SKIP WHEN NULL'
- statement_id: used to distinguish the rows created in table dimension_exceptions

Dimensions in Oracle SQL: usage

Viewing dimension information

```
EXECUTE DBMS_DIMENSION.DESCRIBE_DIMENSION('products_dim');
```

Outline

M2 D&K 2017

5 Querying the DW: ROLAP technology

- SQL Recursive queries
- SQL (OLAP) GROUP BY Extensions
- GROUPING functions
- Analytical functions

Outline

M2 D&K 2017

5 Querying the DW: ROLAP technology

- SQL Recursive queries
 - WITH
- SQL (OLAP) GROUP BY Extensions
- GROUPING functions
- Analytical functions

Database instance for our hierarchical queries:

Employees

EMPLOYEE_ID	LASTNAME	REPORTS_TO	TITLE
1	Davolio	2	Sales Representative
2	Fuller		Vice President, Sales
3	Leverling	2	Sales Representative
4	Peacock	2	Sales Representative
5	Buchanan	2	Sales Manager
6	Suyama	5	Sales Representative
7	King	5	Sales Representative
8	Callahan	2	Inside Sales Coordinator
9	Dodsworth	5	Sales Representative

WITH clause/CTE

CTE=Common Table Expressions

basic syntax

```
WITH subquery_name AS  
(<subquery_expressions>)  
  <query using subquery_name>  
;
```

Supported in Oracle, SQL Server, IBM, MariaDB, PostgresQL.

Query expression defined once can be used multiple times:

- avoid reevaluation overhead
- may avoid incoherent results derived from multiple executions returning different answers
- enable recursive queries

WITH clause (Oracle)

more general syntax

```
WITH subquery_name1 AS ( expressions1 ) search_clause1 cycle_clause1 ,  
     ...  
     subquery_namen AS ( expressionsn ) search_clausen cycle_clausen ,  
     query_expression  
;
```

Column aliases can be introduced after *subquery_name*.

```
WITH  
    reports_to_2 (eid, mgr_id, reportLevel) AS  
(  
    SELECT employee_id, reports_to, 0 reportLevel  
    FROM employees  
    WHERE employee_id = 2  
    UNION ALL  
    SELECT e.employee_id, e.reports_to, reportLevel+1  
    FROM employees e, reports_to_2 r  
    WHERE r.eid = e.reports_to  
)  
SELECT eid, mgr_id, reportLevel  
FROM reports_to_2  
ORDER BY reportLevel, eid;
```

→ recursive query

WITH clause (Oracle) (2)

Result of the query:

EID	MGR_ID	REPORTLEVEL
2		0
1	2	1
3	2	1
4	2	1
5	2	1
8	2	1
6	5	2
7	5	2
9	5	2

Outline

M2 D&K 2017

5 Querying the DW: ROLAP technology

- SQL Recursive queries
- SQL (OLAP) GROUP BY Extensions
 - Motivation
 - CUBE
 - ROLLUP
 - GROUPING SETS
 - Combinations
- GROUPING functions
- Analytical functions

Querying the DW in the relational model

- Analyst formulates OLAP queries on OLAP client.
 - The server must return the answer to queries efficiently.
 - ↪ in the case of ROLAP: simulate dimensional operations.
- ⇒ query language for relational data: SQL.

Querying the DW in the relational model

- Analyst formulates OLAP queries on OLAP client.
 - The server must return the answer to queries efficiently.
 - in the case of ROLAP: simulate dimensional operations.
- ⇒ query language for relational data: SQL.

No specific query language for OLAP but SQL-99 extends SQL-92 with (among others) aggregation functions for OLAP.

OLAP with SQL-92

Why an extension?

Typical OLAP operations:

- DRILL-DOWN: join facts and dimension, GROUP BY
- ROLL-UP: join, GROUP BY
- SLICE, DICE: WHERE clause
- DRILL ACROSS: JOIN, GROUP BY

In theory can be expressed in SQL-92.

OLAP with SQL-92

pivot table for Sales=
all cube aggregations

	Beverages	Produce	Condiments	Total by Quarters
Q1	21	10	18	49
Q2	27	14	11	52
Q3	26	12	35	73
Q4	14	20	47	81
Total by Product	88	56	111	255

Sales		
Quarter Key	Category Key	Amount
Q1	B	21
Q1	P	10
Q1	C	18
Q2	B	27
Q2	P	14
Q2	C	11
Q3	B	26
Q3	P	12
Q3	C	35
Q4	B	14
Q4	P	20
Q4	C	47

OLAP with SQL-92

pivot table for Sales=
all cube aggregations

	Beverages	Produce	Condiments	Total by Quarters
Q1	21	10	18	49
Q2	27	14	11	52
Q3	26	12	35	73
Q4	14	20	47	81
Total by Product	88	56	111	255

SQL query to compute all aggregations from Sales?

Sales aggregations

Quarter Key	Category Key	Amount
Q1	B	21
Q1	P	10
Q1	C	18
Q1	NULL	49
Q2	B	27
Q2	P	14
Q2	C	11
Q2	NULL	52
Q3	B	26
Q3	P	12
Q3	C	35
Q3	NULL	73
Q4	B	14
Q4	P	20
Q4	C	47
Q4	NULL	81
NULL	B	81
NULL	P	56
NULL	C	111
NULL	NULL	255

Shortcomings of SQL-92 for OLAP

Multiple aggregations using UNIONs of GROUP BY:

- painful query formulation
- inefficient

How many granularities for n dimensions (detailed → Top)?

Also hard to express:

- compare the aggregation values from one year to the other
- moving averages
- cumulative aggregations, etc

SQL 99 extensions

Multiple aggregations groupings

Aim: make OLAP queries easier and faster.

Additional aggregation commands from SQL 99:

- **CUBE**: all grouping combinations
- **ROLLUP**: combinations along a hierarchy
- **GROUPING SETS**: specify explicitly a list of grouping combinations requested

Supported in Oracle, SQL Server, IBM DB2, PostgreSQL, MariaDB, *but not MySQL*.

Solution for multiple aggregations in OLAP. For the other issues: additional aggregate functions...

CUBE

```
SELECT city,month, sum(quantity) AS Quantity
FROM Facts F, Time T, Product P, Location L
WHERE P.PID = F.PID
    AND T.TID = F.TID
    AND L.LID = F.LID
    AND P.category = 'DVD'
    AND T.quarter = 'Q1 2010'
GROUP BY CUBE (month, city)
;
```

CITY	MONTH	QUANTITY
		226
Lyon		32
Paris		85
Berlin		67
Stuttgart		42
	fev10	77
Lyon	fev10	12
Paris	fev10	25
Berlin	fev10	25
Stuttgart	fev10	15
	jan10	70
Lyon	jan10	10
Paris	jan10	30
Berlin	jan10	20
Stuttgart	jan10	10
	mar10	79
Lyon	mar10	10
Paris	mar10	30
Berlin	mar10	22
Stuttgart	mar10	17

ROLLUP

```
SELECT quarter, month, SUM(sales) AS sales
FROM Facts F, Time T, Product P
WHERE P.PID = F.PID
  AND T.TID = F.TID
GROUP BY ROLLUP(quarter,month)
;
```

QUARTER	MONTH	SALES
Q1 2010	fev10	422
Q1 2010	jan10	675
Q1 2010	mar10	400
Q1 2010		1497
Q4 2010	dec10	253
Q4 2010		253
		1750

GROUPING SETS

Syntax (basic)

```
... GROUP BY GROUPING SETS (
    (a1,1, a1,2, ..., a1,n1),
    (a2,1, a2,2, ..., a2,n2),
    ...
    (ak,1, ak,2, ..., ak,nk),
);
```

Specifies each requested grouping with a sequence $(a_{i,1}, a_{i,2}, \dots, a_{i,n_i})$.

Equivalent to UNION ALL of GROUP BY queries with NULLs.

GROUPING SET

GROUPING SET

```
SELECT quarter, type, city,
       SUM(sales) AS sales
  FROM Facts F, Time T,
       Product P, Location L
 WHERE P.PID = F.PID
   AND T.TID = F.TID
   AND L.LID = F.LID
   AND T.year = '2010'
GROUP BY GROUPING SETS (
    (quarter, type),
    (quarter, city)
);
```

QUARTER	TYPE	CITY	SALES
Q1 2010	Media		1200
Q1 2010	Livres		297
Q4 2010	Media		253
Q1 2010		Berlin	385
Q1 2010		Paris	385
Q1 2010		Stuttgart	372
Q1 2010		Lyon	355
Q4 2010		Lyon	103
Q4 2010		Berlin	150

GROUPING SETS vs ROLLUP/CUBE

GROUPING SETS can express ROLLUP, CUBE, GROUP BY(a,b).

```
SELECT T.year_id, T.month_id, T.day_id, SUM(amount)
FROM sales S, Time T
WHERE T.day_id= S.day_id
GROUP BY ROLLUP(T.year_id, T.month_id, T.day_id);
```

Express the above query with GROUPING SETS

GROUP BY year_id,month_id with GROUPING SETS?

GROUPING SETS with ROLLUP/CUBE

Complete syntax

```
...GROUP BY GROUPING SETS (
    <attribute sequence1>,
    <attribute sequence2>,
    ...
    <attribute sequencei>,
    ROLLUP <attribute sequence1>,
    ...
    ROLLUP <attribute sequencej>,
    CUBE <attribute sequence1>,
    ...
    CUBE <attribute sequencek>
);
```

Concatenated groupings

Product of groups

```
GROUP BY X, Y=GROUP BY GROUPING SETS (  
    {{ (a1, ..., ai, b1, ..., bj) | <a1, ..., ai> ∈ X, <b1, ..., bj> ∈ Y }}  
);
```

X, Y: lists of attributes, GROUPING SETS and ROLLUP.

Semantics of GROUP BY is
cartesian product of its groups.

Composite columns (b,c)

A collection of columns treated as a unit (i.e., as a single attribute) in the groupings.

Composite column

`ROLLUP(a,(b,c))` is equivalent to `GROUPING SETS((a,b,c),(a),())`

Examples of multiple groups and composite columns

Identify the groups involved

- GROUP BY a, GROUPING SETS ((b), (c,d))
- GROUP BY a, ROLLUP (a, b)
- GROUP BY a, GROUPING SETS ((b), (c), ())
- GROUP BY GROUPING SETS((a,b), (b,c)), GROUPING SETS((d,e),(d),())
- GROUP BY CUBE((a,b),(b,c))

Outline

M2 D&K 2017

5 Querying the DW: ROLAP technology

- SQL Recursive queries
- SQL (OLAP) GROUP BY Extensions
- GROUPING functions
 - GROUPING
 - GROUPING_ID
 - GROUP_ID
- Analytical functions

SQL 99 extensions

Identifying row provenance (groupings)

Additional aggregation commands from SQL 99:

- $\text{GROUPING}(a)^{\dagger\ddagger\$}$: is the row aggregated on attribute a
- $\text{GROUPING_ID}(a_1, \dots, a_n)^{\dagger\ddagger}$: combines multiple $\text{GROUPING}(a_i)$
- $\text{GROUP_ID}()^{\dagger}$: identify redundant groupings from the query

Supported in Oracle[†], SQL Server[‡], IBM DB2[§].

NULLs identification with GROUPING

GROUPING(a)

- returns 1 when the row is a (sub)total for attribute *a*, i.e., when *a* is not part of the group for this row.
- returns 0 otherwise (stored NULL, or non NULL value).

```
SELECT quarter, DECODE(GROUPING(city),1,'multicity',city) city_desc,  
      SUM(sales) sales, GROUPING(city), GROUPING(quarter)  
FROM Facts F, Time T, Location L  
WHERE T.TID = F.TID AND L.LID = F.LID AND L.country='France'  
GROUP BY GROUPING SETS (  
    (quarter),  
    (quarter, city)  
);
```

QUARTER	CITY_DESC	SALES	GROUPING(CITY)	GROUPING(QUARTER)
Q1 2010	Lyon	355	0	0
Q1 2010	Paris	385	0	0
Q1 2010	multicity	740	1	0
Q4 2010	Lyon	103	0	0
Q4 2010	multicity	103	1	0

GROUPING_ID

GROUPING_ID(a_1, \dots, a_n)

- identifies which attributes belong to the grouping
- returns the base 10 value of the bitvector:
GROUPING(a_1) ... GROUPING(a_n)

```
SELECT quarter, city, country, SUM(sales) sales,
       GROUPING_ID(city,quarter,country) GID,
       GROUPING(city) GCT, GROUPING(quarter) GQT , GROUPING(country) GCO
  FROM Facts F, Time T, Location L
 WHERE T.TID = F.TID AND L.LID = F.LID AND L.country='France'
 GROUP BY GROUPING SETS ((country),(quarter, city),(country,city));
```

QUARTER	CITY	COUNTRY	SALES	GID	GCT	GQT	GCO
Q4 2010	Lyon		103	1	0	0	1
Q1 2010	Lyon		355	1	0	0	1
Q1 2010	Paris		385	1	0	0	1
	Lyon	France	458	2	0	1	0
	Paris	France	385	2	0	1	0
		France	843	6	1	1	0

GROUP_ID

GROUP_ID()

- identifies duplicate groupings
- for each grouping, returns 0 for the first set of rows computed according to the grouping, 1 for the second, then 2...

```
SELECT quarter, month,  
       SUM(sales) AS sales, GROUP_ID()  
FROM Facts F, Time T,  
     Product P, Location L  
WHERE P.PID = F.PID  
  AND T.TID = F.TID  
  AND L.LID = F.LID  
GROUP BY quarter, ROLLUP(quarter,month)  
;
```

QUARTER	MONTH	SALES	GROUP_ID()
Q1 2010	fev10	422	0
Q1 2010	jan10	675	0
Q1 2010	mar10	400	0
Q4 2010	dec10	253	0
Q1 2010		1497	0
Q4 2010		253	0
Q1 2010		1497	1
Q4 2010		253	1

Eliminate duplicate groupings.

GROUP_ID

GROUP_ID()

- identifies duplicate groupings
- for each grouping, returns 0 for the first set of rows computed according to the grouping, 1 for the second, then 2...

```
SELECT quarter, month,  
       SUM(sales) AS sales, GROUP_ID()  
FROM Facts F, Time T,  
     Product P, Location L  
WHERE P.PID = F.PID  
      AND T.TID = F.TID  
      AND L.LID = F.LID  
GROUP BY quarter, ROLLUP(quarter,month)  
;
```

QUARTER	MONTH	SALES	GROUP_ID()
Q1 2010	fev10	422	0
Q1 2010	jan10	675	0
Q1 2010	mar10	400	0
Q4 2010	dec10	253	0
Q1 2010		1497	0
Q4 2010		253	0
Q1 2010		1497	1
Q4 2010		253	1

Eliminate duplicate groupings.

Outline

M2 D&K 2017

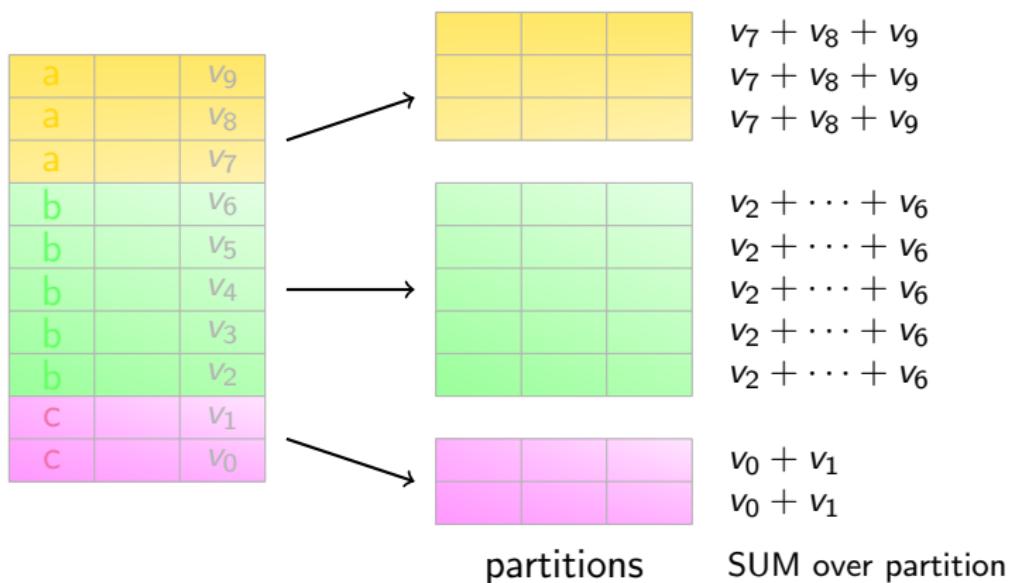
5 Querying the DW: ROLAP technology

- SQL Recursive queries
- SQL (OLAP) GROUP BY Extensions
- GROUPING functions
- Analytical functions
 - Windows
 - Ranking and Aggregation function
 - PIVOT

Window partitioning

Window partitioning

partitions input tuples, then aggregate independently each partition. Each original row is preserved; aggregated value comes as additional column.



Ex: add to each row the total sales (SUM) within each partition

Supported by Oracle, SQL Server, DB2, PostgreSQL, MariaDB(dev) but not MySQL.

Window partitioning

window_function OVER ([PARTITION BY attribute_sequence])

partitions tuples, then aggregate independently each partition.

Contribution to sales per city and period

```
SELECT quarter, date, city, sales,
       SUM(sales) OVER(PARTITION BY quarter) AS SALES_QT
  FROM Facts F, Time T, Product P, Location L
 WHERE P.PID = F.PID
   AND T.TID = F.TID
   AND L.LID = F.LID;
```

QUARTER	DATE	CITY	SALES	SALES_QT
Q1 2010	01Jan10	Berlin	120	1497
Q1 2010	04Feb10	Berlin	120	1497
Q1 2010	20Mar10	Berlin	145	1497
Q1 2010	01Jan10	Lyon	355	1497
Q1 2010	10Mar10	Paris	385	1497
Q1 2010	10Jan10	Stuttgart	372	1497
Q4 2010	18Nov10	Berlin	50	253
Q4 2010	19Nov10	Berlin	100	253
Q4 2010	16Dec10	Lyon	103	253

Window partitioning

window_function OVER ([PARTITION BY attribute_sequence])

combined with GROUP BY:

Contribution to sales per city and period

```
SELECT quarter, city,
       SUM(SUM(sales)) OVER(PARTITION BY quarter) AS SALES_QT,
       SUM(sales) AS salesC
  FROM Facts F, Time T, Product P, Location L
 WHERE P.PID = F.PID
   AND T.TID = F.TID
   AND L.LID = F.LID
 GROUP BY quarter, city;
```

QUARTER	CITY	SALES_QT	SALESC
Q1 2010	Berlin	1497	385
Q1 2010	Lyon	1497	355
Q1 2010	Paris	1497	385
Q1 2010	Stuttgart	1497	372
Q4 2010	Berlin	253	150
Q4 2010	Lyon	253	103

Window ordering

window_function OVER ([PARTITION BY ...] [ORDER BY ...])

- orders rows within each partition
- aggregation restricted to preceding rows within the partition. 

Rank of city w.r.t. sales, within trimester

```
SELECT quarter, city,
       RANK() OVER(PARTITION BY quarter ORDER BY SUM(sales)) AS RG,
       SUM(sales) AS sales
  FROM Facts F, Time T, Product P, Location L
 WHERE P.PID = F.PID AND T.TID = F.TID AND L.LID = F.LID
 GROUP BY quarter, city;
```

QUARTER	CITY	RG	SALES
Q1 2010	Lyon	1	355
Q1 2010	Stuttgart	2	372
Q1 2010	Berlin	3	385
Q1 2010	Paris	3	385
Q4 2010	Lyon	1	103
Q4 2010	Berlin	2	150

Window framing: ROWS

window_function OVER ([PARTITION BY ...] [ORDER BY ...] [window_frame])

- framing clause limits the scope of aggregation within each partition
- In ROW mode, frame is specified as nb of rows around current row.

Average sales of the city over last 3 months

```
SELECT T.TID AS T, month, city,
       AVG(SUM(sales)) OVER(
         PARTITION BY city
         ORDER BY T.TID
         ROWS 2 PRECEDING) sales3,
       SUM(sales) AS sales
  FROM Facts F, Time T,
       Product P, Location L
 WHERE P.PID = F.PID
   AND T.TID = F.TID
   AND L.LID = F.LID
 GROUP BY month, T.TID, city
 ORDER BY city, T.TID;
```

T	MONTH	CITY	SALES3	SALES
1	jan10	Berlin	163	163
2	fev10	Berlin	142.5	122
3	mar10	Berlin	128.333333	100
4	dec10	Berlin	124	150
1	jan10	Lyon	155	155
2	fev10	Lyon	127.5	100
3	mar10	Lyon	118.333333	100
4	dec10	Lyon	101	103
1	jan10	Paris	185	185
2	fev10	Paris	142.5	100
3	mar10	Paris	128.333333	100
1	jan10	Stuttgart	172	172
2	fev10	Stuttgart	136	100
3	mar10	Stuttgart	124	100

Window framing: RANGE

window_function OVER ([PARTITION BY ...] [ORDER BY ...] [window_frame])

- RANGE mode: frame specified by de/in-crementing ORDER BY expr. of current row. \hookrightarrow ORDER BY must be single expr. (ex: numeric or date).
- called “logical offset” as opposed to physical.

Counting cities with sales amount between 90% and 100% of current city, per trimester

```
SELECT quarter, city,
       COUNT(*) OVER(
         PARTITION BY quarter
         ORDER BY SUM(sales)
         RANGE 0.1*SUM(sales) PRECEDING) NB,
       SUM(sales) AS sales
  FROM Facts F, Time T,
       Product P, Location L
 WHERE P.PID = F.PID
   AND T.TID = F.TID
   AND L.LID = F.LID
 GROUP BY quarter, city;
```

QUARTER	CITY	NB	SALES
Q1 2010	Lyon	1	355
Q1 2010	Stuttgart	2	372
Q1 2010	Berlin	4	385
Q1 2010	Paris	4	385
Q4 2010	Lyon	1	103
Q4 2010	Berlin	1	150

Scope of aggregation

- **OVER():** all tuples
- **OVER(PARTITION BY):** all tuples in same partition
- **OVER(ORDER BY):** all tuples smaller than current tuple,
=RANGE UNBOUNDED PRECEDING
- **ROWS UNBOUNDED PRECEDING:** all preceding tuples in same partition
- **ROWS k PRECEDING:** k preceding tuples in same partition
- **UNBOUNDED FOLLOWING**
 - k FOLLOWING**
 - CURRENT ROW**
 - BETWEEN**

ROWS BETWEEN 2 PRECEDING AND CURRENT ROW = ROWS 2 PRECEDING
- **RANGE:** like ROWS but allow to define scope w.r.t. value instead of rank (number of lines)



ROWS, RANGE require order be defined.

Aggregation functions

- Analytical functions (set semantics \Rightarrow relational):

SUM, COUNT, MIN, MAX, AVG, EVERY, ANY,
STDDEV_POP, VAR_SAMP, PERCENTILE_CONT...

- Ranking functions (w.r.t. some order):

RANK, DENSE_RANK, PERCENT_RANK, CUME_DIST,
ROW_NUMBER, NTILE, LEAD, LAG, FIRST_VALUE...

 DISTINCT aggregates and most ranking functions (but not FIRST_VALUE...) are not affected by framing clause.

Ranking functions

ROW_NUMBER: ties ranked arbitrarily

RANK vs DENSE_RANK: no value gap after ties

```
SELECT month, city, SUM(sales) AS sales,
       RANK() OVER (ORDER BY SUM(sales) DESC) RK,
       DENSE_RANK() OVER (ORDER BY SUM(sales) DESC) DENSE_RK,
       ROW_NUMBER() OVER (ORDER BY SUM(sales) DESC) ROW_NUM
  FROM Facts F, Time T, Location L
 WHERE T.TID = F.TID AND L.LID = F.LID
 GROUP BY month,city;
```

MONTH	CITY	SALES	RK	DENSE_RK	ROW_NUM
jan10	Paris	185	1	1	1
jan10	Berlin	172	2	2	2
jan10	Stuttgart	172	2	2	3
jan10	Lyon	155	4	3	4
dec10	Berlin	150	5	4	5
...					

Reporting functions

LAG/LEAD (*value_expr* [, *offset*] [, *default*]) [RESPECT NULLS | IGNORE NULLS]
OVER ([*query_partition_clause*] *order_by_clause*)

```
SELECT month,city, SUM(sales) AS sales,  
       LAG(SUM(sales),1,-1) OVER (  
           PARTITION by city  
           ORDER BY to_date(month,'monYY'))  
           AS sales_prev  
FROM Facts F, Time T, Location L  
WHERE T.TID = F.TID AND L.LID = F.LID  
GROUP BY month,city;
```

MONTH	CITY	SALES	SALES_PREV
jan10	Berlin	163	-1
feb10	Berlin	122	163
mar10	Berlin	100	122
dec10	Berlin	150	100
jan10	Lyon	155	-1
feb10	Lyon	100	155
mar10	Lyon	100	100
dec10	Lyon	103	100
jan10	Paris	185	-1
feb10	Paris	100	185
mar10	Paris	100	100
jan10	Stuttgart	172	-1
feb10	Stuttgart	100	172
mar10	Stuttgart	100	100

LEAD : same but offset follows instead of precede current row

Reporting functions

FIRST_VALUE, LAST_VALUE : select FIRST/LAST row of aggregation scope.

```
SELECT month,city, SUM(sales) AS sales,
       FIRST_VALUE(SUM(sales)) OVER (
           PARTITION by month
           ORDER BY SUM(sales) DESC)
       AS sales_max
  FROM Facts F, Time T, Location L
 WHERE T.TID = F.TID AND L.LID = F.LID
 GROUP BY month,city;
```

MONTH	CITY	SALES	SALES_MAX
dec10	Berlin	150	150
dec10	Lyon	103	150
feb10	Berlin	122	122
feb10	Lyon	100	122
feb10	Paris	100	122
feb10	Stuttgart	100	122
jan10	Paris	185	185
jan10	Stuttgart	172	185
jan10	Berlin	163	185
jan10	Lyon	155	185
mar10	Lyon	100	100
mar10	Paris	100	100
mar10	Berlin	100	100
mar10	Stuttgart	100	100

⚠ What result if we replace FIRST_VALUE with LAST_VALUE?

Pivoting

Principle: on fact table with sales measure, *GROUP BY quarter,city* would return 1 line per city. Instead, *pivot* query returns a single line per *quarter* with one column per city.

```
SELECT * FROM
  (SELECT quarter, city, sales
   FROM SALES_TRIM_CITY
  )
PIVOT (SUM(sales)
      FOR city IN ('Paris' AS lutetia, 'Lyon' AS Lugdunum)
)
;
```

QUARTER	CITY	SALES
Q1 2010	Berlin	385
Q1 2010	Paris	385
Q1 2010	Stuttgart	372
Q1 2010	Lyon	355
Q4 2010	Lyon	103
Q4 2010	Berlin	150



QUARTER	LUTETIA	LUGDUNUM
Q1 2010	385	355
Q4 2010		103

Supported in Oracle, SQL Server.

References

Oracle DataWarehousing guide (inspired the whole section):

http://docs.oracle.com/cd/E11882_01/server.112/e25554/analysis.htm

<http://sqlpro.developpez.com/article/olap-clause-window/>

http://www.dba-oracle.com/t_advanced_sql_windowing_clause.htm

<http://msdn.microsoft.com/fr-fr/library/ms189461.aspx>

<http://www.vldb.org/pvldb/vol8/p1058-leis.pdf>

Outline

M2 D&K 2017

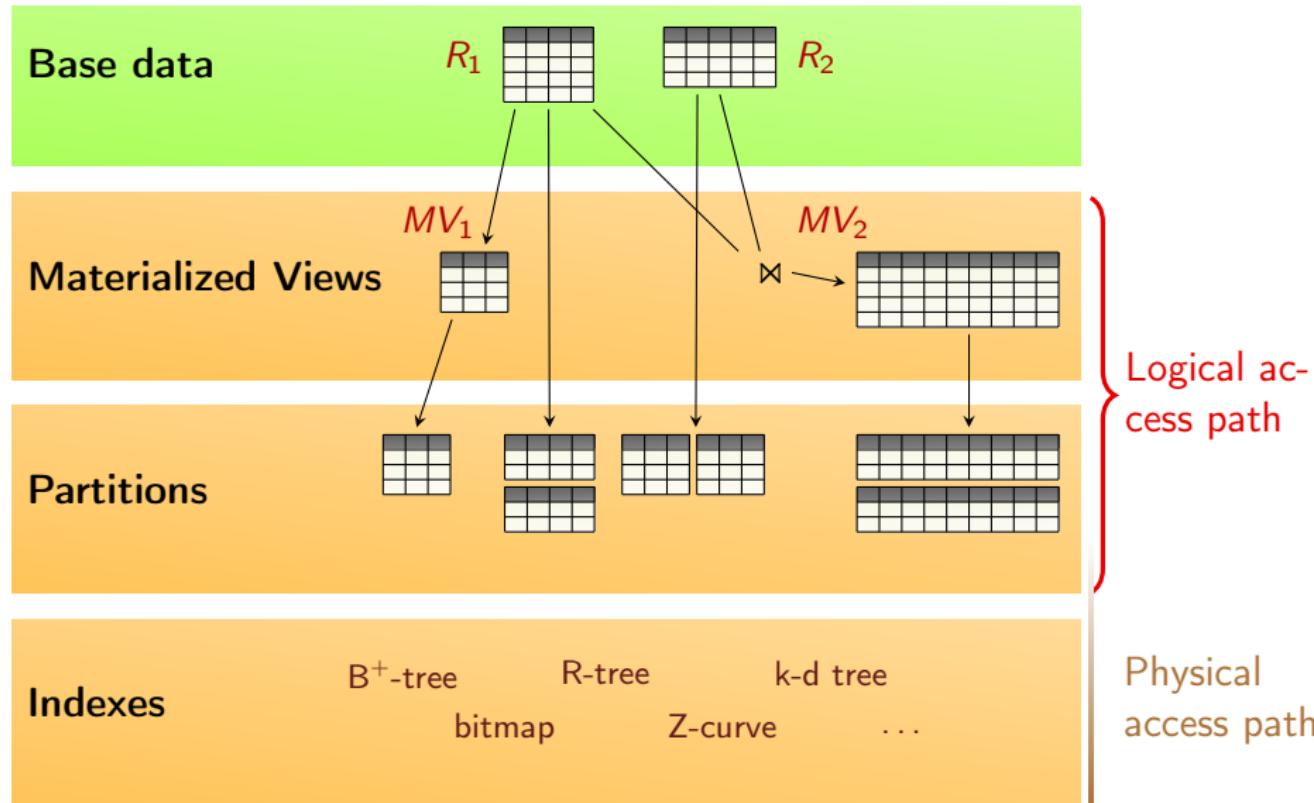
⑥ Physical design and query optimization

- Partitioning
- Indexes
- Clustering

3 levels of design

Conceptual design	User-oriented description, independant of implementation <i>E-R, UML</i>
Logical design	Logical description, independant of DBMS <i>Relational model: table schema</i>
Physical design	Actual database structures <i>materialized views, partitions, indexes</i>

Accessing data



: logical schema (ANSI SPARC architecture)



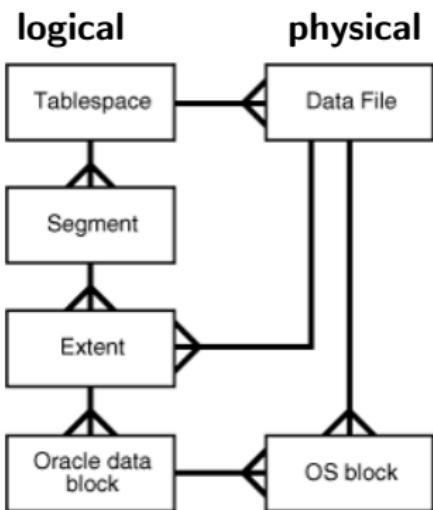
: physical schema (ANSI SPARC architecture)

Physical storage in DBMS (reminder)

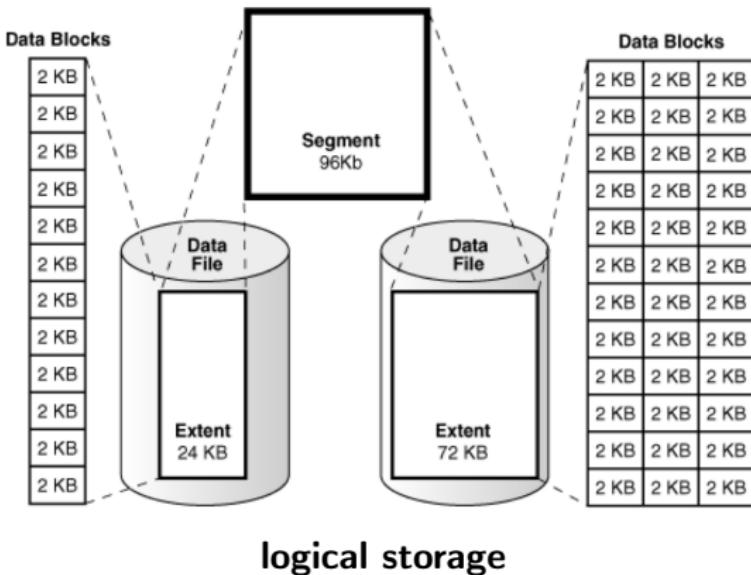
- physical address of record provided by ROWID
- Records stored in pages=blocks.
- Read/Write operations require the corresponding data be brought in buffer (main memory).
- DBMS must minimize pages I/O for performance.

larger block size: more tuples edited per page I/O, but fewer pages in main-memory.

Physical storage: Oracle (1)

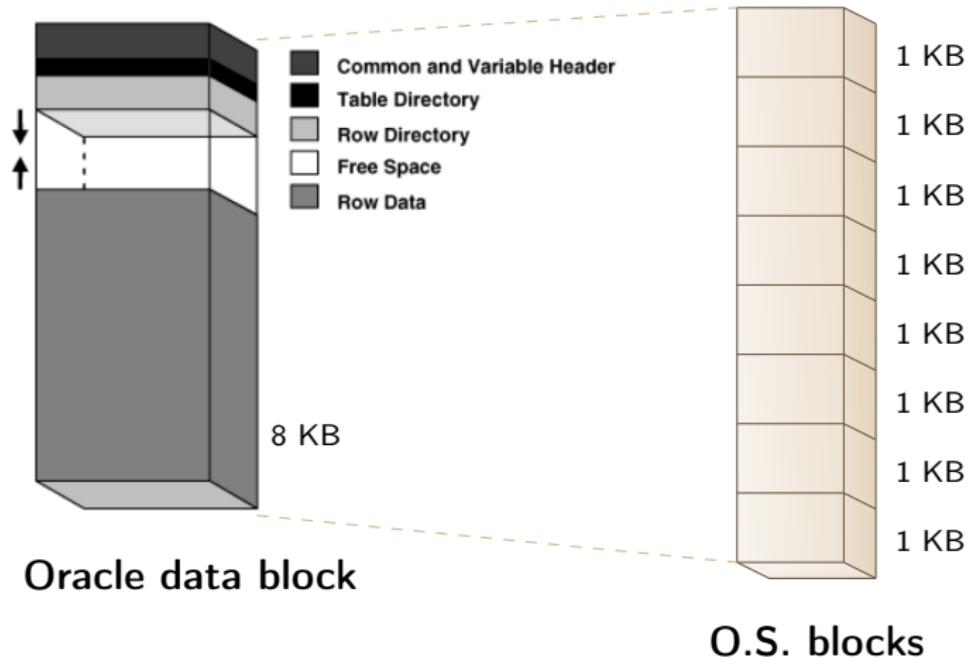


E-R diagram of
storage structures



[Oracle Database Concepts]

Physical storage: Oracle (2)



References

Oracle Database Concepts

<https://docs.oracle.com/database/121/CNCPT/logical.htm>

https://en.wikibooks.org/wiki/Oracle_and_DB2,_Comparison_and_Compatibility/Storage_Model/Physical_Storage/Summary

ftp:
[//ftp.software.ibm.com/software/data/db2/9/labchats/20110331-slides.pdf](http://ftp.software.ibm.com/software/data/db2/9/labchats/20110331-slides.pdf)
(a bit outdated: 2011...)

<http://www.postgresql.org/docs/9.4/static/storage-page-layout.html>

Outline

M2 D&K 2017

⑥ Physical design and query optimization

- Partitioning

- Partitioning concepts
- Vertical partitioning
- Horizontal partitioning: creating partitions
- Horizontal partitioning: maintenance
- Horizontal partitioning: benefits

- Indexes

- Clustering

Partitioning

Principle: divide data (table, index) into pieces that can be manipulated independently.

- ↪ a key feature in datawarehouses!
- several kinds of partitioning (depending on vendors too)
- choice of parameters may be crucial for performance
- transparency (hopefully): queries need not be rewritten

Assets:

- ✓ more indexing opportunities
- ✓ easier reorganization/update (monitoring)

Weaknesses:

- ✗ performance overhead for queries spanning multiple partitions
- ✗ more tables/complex schema

Partitioning

Principle: divide data (table, index) into pieces that can be manipulated independently.

2 ways to partition a table:

- *vertical partitioning*: split the columns
- *horizontal partitioning*: splits the rows

We only detail partitioning in the DBMS, but also possible to partition data at the application level, or hardware.

Vertical partitioning

Vertical partitioning/Row splitting

splits the attribute of the table into groups. Each group is stored in a separate table.

We already saw a particular (but frequent) case of vertical partitioning:

id	price	amount	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	
3	30	1	DS27W	VISA	



id	price	amount
1	50	4
2	60	2
3	30	1

id	code	form	...
1	AX03F	CASH	
2	VF67D	VISA	
3	DS27W	VISA	

Vertical partitioning

Vertical partitioning/Row splitting

splits the attribute of the table into groups. Each group is stored in a separate table.

We already saw a particular (but frequent) case of vertical partitioning:
mini-dimension

id	price	amount	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	
3	30	1	DS27W	VISA	



id	price	amount
1	50	4
2	60	2
3	30	1

id	code	form	...
1	AX03F	CASH	
2	VF67D	VISA	
3	DS27W	VISA	

Vertical partitioning

- more tables, but fewer columns in each
- different physical devices can be used for each partition
- a view across all partitions allows to restore original rows
- used to store large (BLOBs, LONGs) or seldom-used attributes in separate table (e.g., cheaper storage media)

Assets:

- ✓ smaller records: more can be loaded *in-memory*
- ✓ limit reorganization on frequently changing attributes
- ✓ each partition can be optimized independently
- ✓ contention (physical storage), parallelism

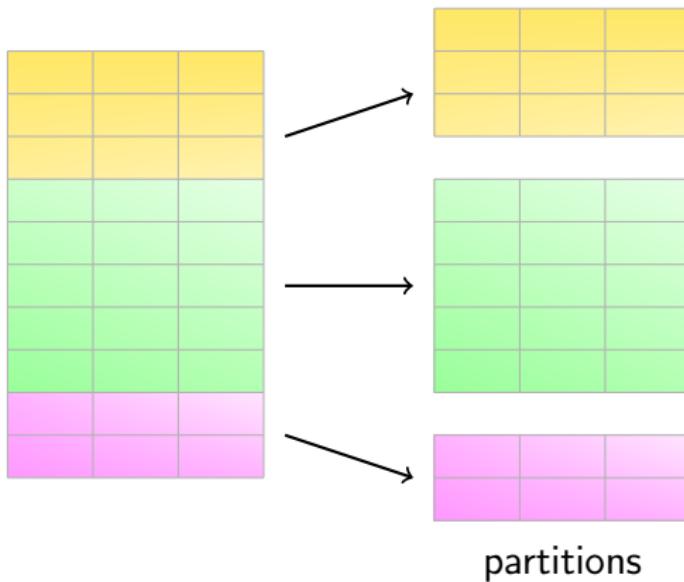
Weaknesses:

- ✗ destroys semantic unit: record
- ✗ joining tables through non-materialized view degrades performance

Horizontal partitioning: principle

Horizontal partitioning

splits the rows of the table into groups based on attribute value. Each group is stored in a separate table.



Featured in all major DBMS: Oracle, IBM DB2, SQL Server, PostgreSQL, Mysql, MariaDB...

- allows to handle very large relations
- may simplify maintenance
- can be used to filter partitions for query optimization (partition pruning)
- allows parallelism

Horizontal partitioning

Horizontal partitioning

splits the rows of the table into groups based on attribute value. Each group is stored in a separate table.

id	price	amount	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	
3	30	1	DS27W	VISA	
4	20	3	AP11Z	CASH	



partitioning key

id	price	amount	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	

id	price	amount	code	form	...
3	30	1	DS27W	VISA	
4	20	3	AP11Z	CASH	

Horizontal partitioning

splits the rows of the table into groups based on attribute value. Each group is stored in a separate table.

id	price	amount	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	
3	30	1	DS27W	VISA	
4	20	3	AP11Z	CASH	



id	price	amount	code	form	...
1	50	4	AX03F	CASH	
4	20	3	AP11Z	CASH	

id	price	amount	code	form	...
3	30	1	DS27W	VISA	
2	60	2	VF67D	VISA	

Horizontal partitioning

... each tuple is assigned to exactly one partition depending on value of partitioning key(s).

Oracle syntax (sketch)

```
PARTITION BY <partition_type> (attribute list) ...  
                                partitioning key
```

Horizontal partitioning

... each tuple is assigned to exactly one partition depending on value of partitioning key(s).

Oracle syntax (sketch)

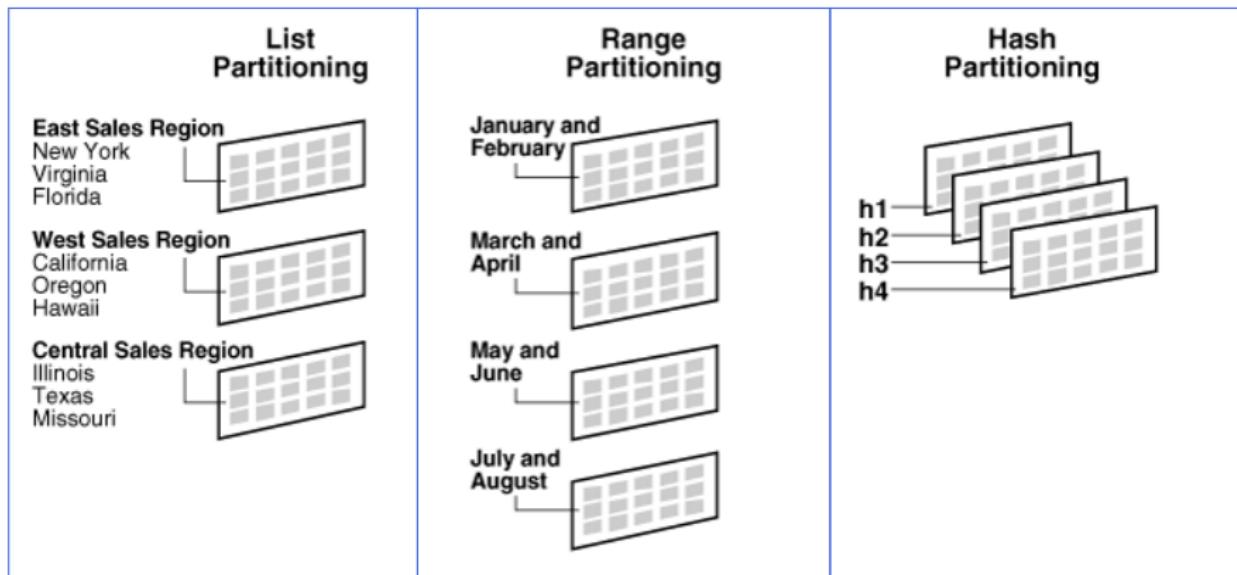
```
PARTITION BY <partition_type> (attribute list) ...  
                                partitioning key
```



in multicolumn keys, column $k + 1$ only breaks ties on columns $(1, \dots, k)$

Multiple horizontal partitioning options: Oracle

Single level partitioning

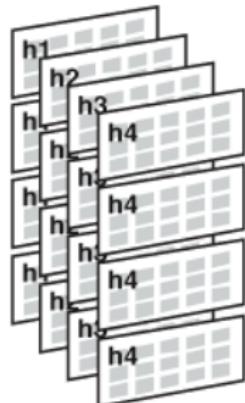


[Data Warehousing Guide, Oracle]

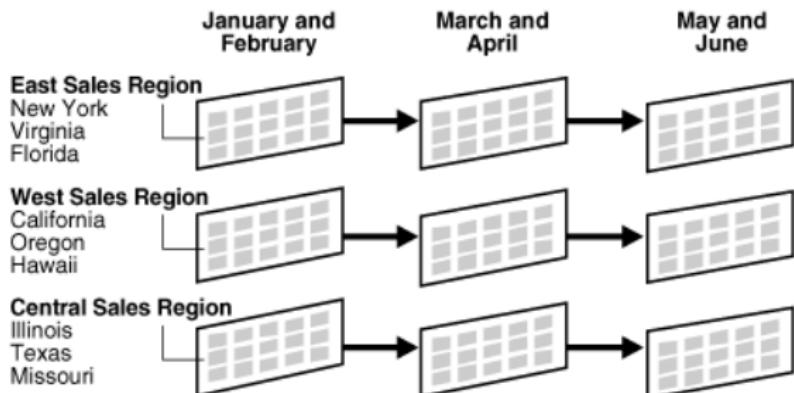
Composite partitioning examples: Oracle

Composite partitioning

Composite Partitioning
Range-Hash



Composite Partitioning
Range - List



[Data Warehousing Guide, Oracle]

RANGE partitioning (Oracle)

PARTITION BY RANGE(column_list) ... VALUES LESS THAN (value list)

Partitions are specified by upper bound (exclusive).

```
CREATE TABLE sales_range(sales_id int PRIMARY KEY, salesman_id NUMBER(5),  
    salesman_name VARCHAR2(30), sales_amount NUMBER(10),sales_date DATE)  
PARTITION BY RANGE(sales_date)  
    (PARTITION t21 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),  
     PARTITION t22 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),  
     PARTITION t23 VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY')),  
     PARTITION t24 VALUES LESS THAN(TO_DATE('05/01/2000','DD/MM/YYYY')));
```

MAXVALUE can be used to specify the top range, otherwise: implicit integrity constraint on the table.

[Data Warehousing Guide, Oracle]

HASH partitioning (Oracle)

```
CREATE TABLE sales_hash
(salesman_id NUMBER(5),salesman_name VARCHAR2(30),
sales_amount NUMBER(10),week_no NUMBER(2))
PARTITION BY HASH(salesman_id)
PARTITIONS 4; -- number of partitions
```

For optimal distribution:

- choose adequate partitioning key
(e.g., column(s) that are almost unique)
- take power of 2 as number of partitions (subpartitions)

LIST partitioning (Oracle)

```
CREATE TABLE sales_list (salesman_id NUMBER(5),
    sales_state VARCHAR2(20), sales_amount NUMBER(10))
PARTITION BY LIST(sales_state) -- single attribute: no composite column
(PARTITION sales_west VALUES('California', 'Hawaii'),
PARTITION sales_east VALUES ('New York', 'Virginia', 'Florida'),
PARTITION sales_other VALUES(DEFAULT)); -- (optional)
```



Does not support multi-column partition keys.

Additional partitioning options (Oracle)

Interval partitioning

```
CREATE TABLE sales_interv(salesman_id NUMBER(5),salesman_name VARCHAR2(30),
  sales_amount NUMBER(10),sales_date DATE)
PARTITION BY RANGE(sales_date)
INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))
(PARTITION t21 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),
-- a first range partition must be declared
PARTITION t22 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),
PARTITION t23 VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY')));
-- partitions after 4/01/2000 with interval width of 1 month
```

Partition by reference

```
CREATE TABLE sales_item(sales_id int NOT NULL,
  order_id int NOT NULL, desc VARCHAR2(10),
  CONSTRAINT sales_item_fk
  FOREIGN KEY(sales_id) REFERENCES sales_range(sales_id)
)
PARTITION BY REFERENCE(sales_item_fk)
-- requires named referential constraint toward partitioned table
```

Additional partitioning options (Oracle)

Partitioning on virtual column

```
CREATE TABLE sales_virt
(
    sale_id NUMBER(5)
    , quantity_sold NUMBER(6) NOT NULL
    , unit_price NUMBER(8,2) NOT NULL
    , total_amount AS (quantity_sold*unit_price) -- cannot use PLSQL
)
PARTITION BY HASH(total_amount)
PARTITIONS 4;
```

COMPOSITE partitioning (Oracle)

```
CREATE TABLE sales_range_hash(s_productid NUMBER, s_saledate DATE,  
    s_custid NUMBER, s_totalprice NUMBER)  
PARTITION BY RANGE (s_saledate)  
SUBPARTITION BY HASH (s_productid) SUBPARTITIONS 8  
    (PARTITION sal99q1 VALUES LESS THAN (TO_DATE('01-APR-1999', 'DD-MON-YYYY')),  
     PARTITION sal99q4 VALUES LESS THAN (TO_DATE('01-JAN-2000', 'DD-MON-YYYY')));
```

Partitioning: which and when?

- as a general rule, consider partitioning in Oracle if:
 - table > 2Gb, more than 1M rows
 - historic data are read-only
 - multiple media storages
- RANGE partitioning: typically (not exclusively) used on temporal attributes
- HASH partitioning: to balance partition size, typical for non-temporal attributes, or to distribute data on physical storage medias
- COMPOSITE partitioning: depending on applications, if high degree of parallelism required

 *unbalanced partitions tend to degrade performance*

Horizontal partitioning: updates

ROW MOVEMENT clause

Oracle syntax

```
CREATE TABLE...  
PARTITION BY...  
DISABLE/ENABLE ROW MOVEMENT
```

- *row movement enabled*: updating the key may cause a row to move to another partition
- *row movement disabled (default)*: update fails if it would result in row migration

Managing partitions:maintenance

Operations on partitions (or subpartitions)

- ADD: introduce a new partition (or subpartition)
 - DROP: removes the partition and delete its content
 - TRUNCATE: delete content of a partition
-
- MERGE: merges 2 partitions into a single one
 - SPLIT: redistribute content of partition into 2 new ones
-
- EXCHANGE: move partition to/from table

and also:

MODIFY, MOVE, RENAME, COALESCE

Add partition

ALTER TABLE <tablename> ADD PARTITION ...

```
ALTER TABLE sales_range  
  ADD PARTITION jan00 VALUES LESS THAN ('01-FEB-2000')  
  TABLESPACE tsx;
```

```
ALTER TABLE sales_range_hash MODIFY PARTITION sal99q1  
  ADD SUBPARTITION subpart9 TABLESPACE ts99q1;  
-- the rows within partition sal99q1 are re-hashed
```



operations depend on partition type.

DROP partition

ALTER TABLE <tablename> DROP PARTITION ...

Fastest way to delete large volumes of data.

```
ALTER TABLE sales_range DROP PARTITION jan00;
-- take special care for (global) indexes and integrity constraints:

-- indexes: add 'UPDATE INDEXES' clause
-- or 'ALTER INDEX sales_idx REBUILD'
-- or first 'DELETE FROM sales_range PARTITION (jan00)' before DROP

-- integrity constraints: disable them, or first DELETE
```

Maintaining indexes during DROP may degrade performance.

TRUNCATE: similar to DROP, but only rows are deleted;
the partition (emptied) remains.



operations depend on partition type.

SPLIT/MERGE partition(s)

ALTER TABLE <tablename> SPLIT PARTITION ...

```
ALTER TABLE sales_range
SPLIT PARTITION t21 AT (TO_DATE('01/01/2000','DD/MM/YYYY'))
INTO (PARTITION t21a, Partition t21b);
```

ALTER TABLE <tablename> MERGE PARTITIONS ...

```
ALTER TABLE sales_range
MERGE PARTITIONS t21a, t21b INTO PARTITION t21;
```



operations depend on partition type.

Partition exchange

Oracle syntax

```
ALTER TABLE <tname>
EXCHANGE PARTITION <pname>
WITH TABLE <tname2>
[INCLUDING/EXCLUDING INDEXES] - - - -> exchange index(es)
[WITH/WITHOUT VALIDATION] - - - -> check rows suitable for partition
```

- Swaps a table and a partition.
- table and partition must have same schema, FK constraint, indexes
- extremely useful in ETL.

Typically, organize new data; indexes, constraints... in dedicated table, then integrate new data using partition exchange.

```
CREATE TABLE sales_apr_src(sales_id int PRIMARY KEY, salesman_id NUMBER(5),
salesman_name VARCHAR2(30), sales_amount NUMBER(10),sales_date DATE);

INSERT INTO sales_apr (1,2,'T. Elliot',23,TO_DATE('04/21/2000','DD/MM/YYYY'));

ALTER TABLE sales_range EXCHANGE PARTITION t24 WITH TABLE sales_apr;
```

Partitioning benefits

(Reminder)

- allows to handle very large relations
- allows parallelism
- may simplify maintenance
- allows to filter partitions for query optimization (partition pruning)

Partitioning benefits

(Reminder)

- allows to handle very large relations
- allows parallelism
- may simplify maintenance ✓ (ETL,archiving...)
- allows to filter partitions for query optimization (partition pruning)

Partition pruning

Principle: "*Do not scan partitions where there can be no matching values*"
when the query filters values based on partitioning column.

Optimizer selects relevant partitions, scan partition instead of whole table.

Predicates supported:

RANGE, LIST *IN, =, LIKE, range (<,...)*

HASH *IN, =*

Oracle distinguishes

- static partition pruning: optimizer determines partition at compile time (no subquery, static predicate...)
- dynamic partition pruning: for statements that use subqueries... .

Partition-wise joins

Parallelism within a query (principle): *split the query in multiple subqueries over distinct parts of the data; process subqueries in parallel.*

Aim: speedup join processing by minimizing data exchange between parallel execution servers (reduces both communication and memory).

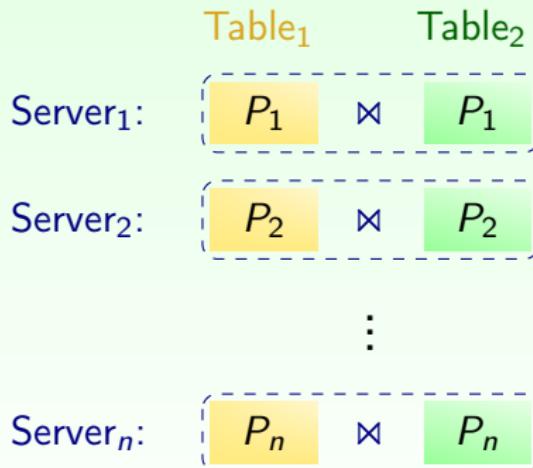
Partition-wise joins can be:

- "*full*" : the 2 tables are partitioned on join key
- "*partial*" : 1 of the tables must be partitioned on join key

Full partitions-wise joins

Joined tables must be partitioned on join key, with same partition criterias (for hash: same # partitions).

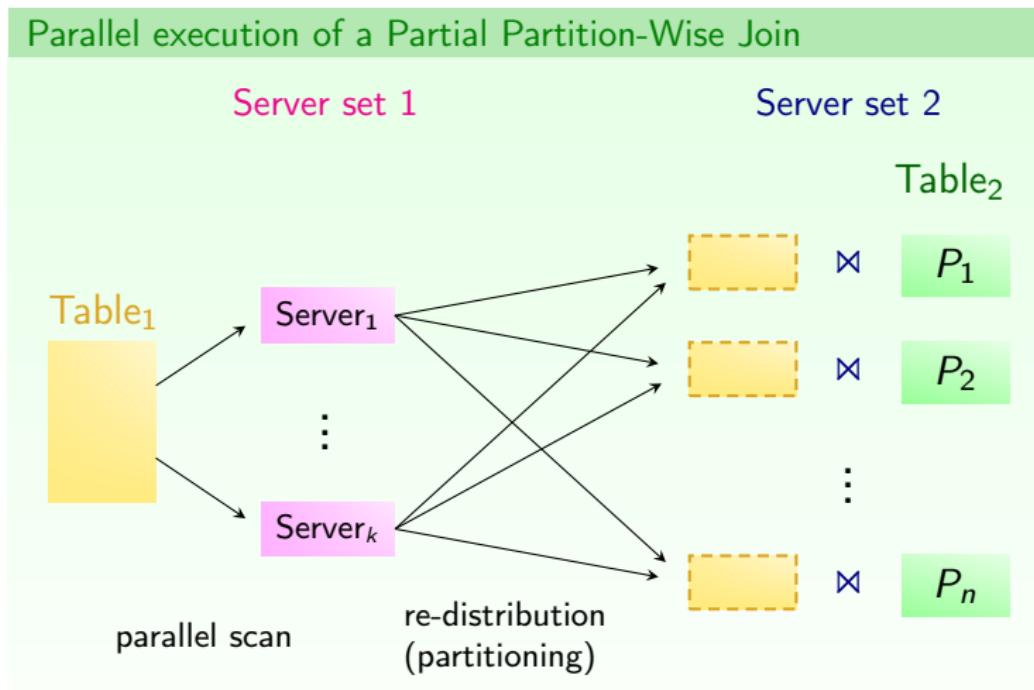
Parallel execution of a Full Partition-Wise Join



- Can be processed serially or in parallel
- Reference partitioning is an easy way to enable full partition-wise joins.

Partial partition-wise joins

Second table needs not be partitioned on join key, as it will be dynamically (re)partitioned based on the reference table partition.



Can be processed in parallel only.

References

Vendors' doc:

VLDB and Partitioning Guide, Oracle (inspired the whole section):

<https://docs.oracle.com/database/121/VLDBG/title.htm>

<https://msdn.microsoft.com/en-us/library/ms190787.aspx>

http://www-01.ibm.com/support/knowledgecenter/SSEPGG_9.7.0/com.ibm.db2.luw.admin.partition.doc/doc/c0021560.html

<http://www.postgresql.org/docs/9.4/static/ddl-partitioning.html>

<https://mariadb.com/kb/en/mariadb/using-connect-partitioning-and-sharding/>

Outline

M2 D&K 2017

⑥ Physical design and query optimization

- Partitioning

- Indexes

- Indexing
- B-trees
- Bitmap Indexes
- Bitmap Join Indexes
- Joins and partitioning

- Clustering

Indexes in DBMS

Aim of an index: provide fast access to the data needed by the query.

Ex:

```
SELECT *  
FROM Employee  
WHERE EmployeeKey = 1234;
```

- With index on EmployeeKey: 1 disk access.
- Without: scan whole Employee table

Drawback: updates on indexed attribute require index update.
⇒ too many indexes degrade performance.

Index taxonomy

- *single-column* vs multicolmn
- *clustered* vs non-clustered: in a clustered index, records are physically ordered according to the index key.
At most 1 clustered index, but several non-clustered indexes are possible.
- *unique* vs non-unique: in a unique index, only one record per key value whereas non-unique allow duplicates
- *dense* vs sparse: a dense index has one entry per data record.

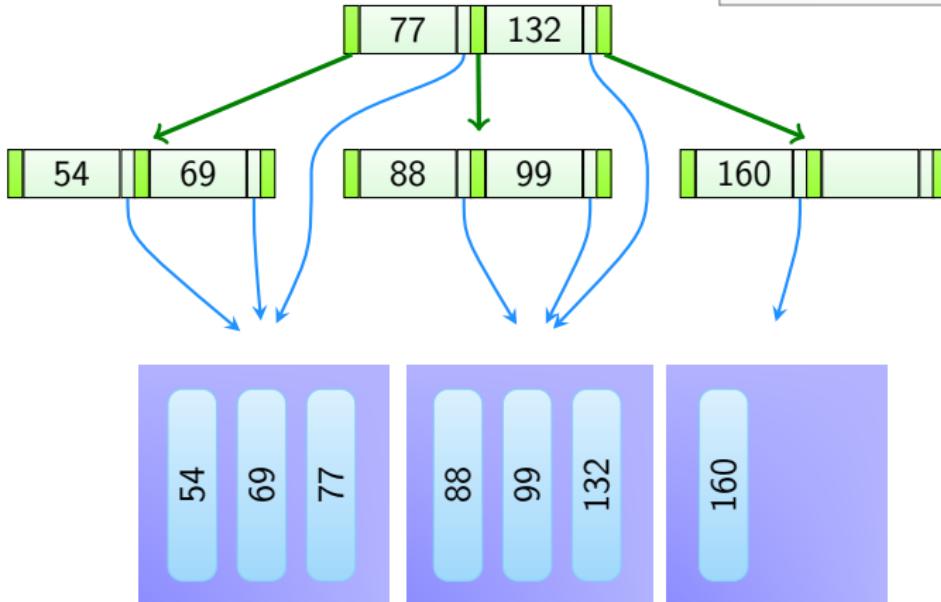
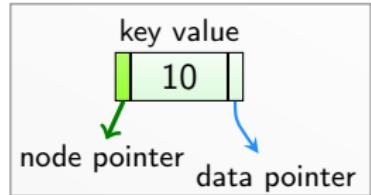
Indexes and partitions

- *local index*: index partitions follow the partitions of the database: 1-1 correspondance between table and index partitions. Optimizes maintenance. Most common in DW.
- *global partitioned index* : index partitions do not correspond to table partitions. More common in OLTP.
- *global non-partitioned index*: index is not partitioned. Mostly used to enforce unique key constraints. But one can also rely on ETL for that.

B-trees

Bayer, McCreight '72

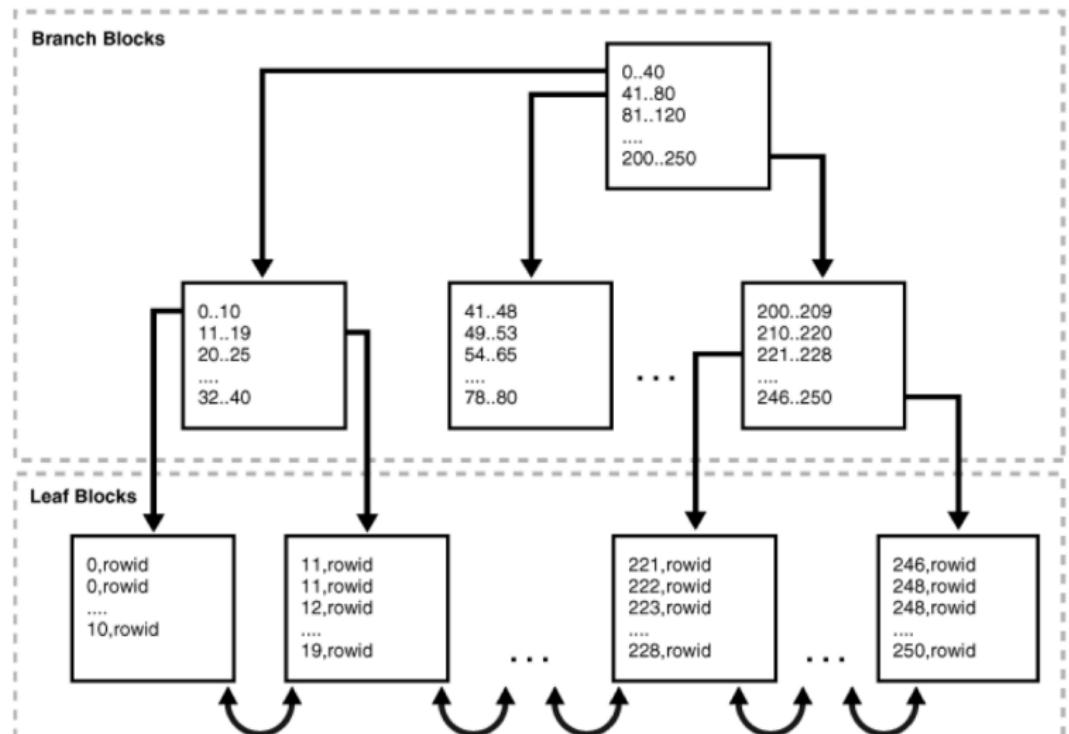
The typical index in operational DB is a B-tree.



Between m and $2m$ keys per node (root may have less).
Leaves belong to same level (balanced tree).

B-trees: a "B-tree" index from Oracle

Figure 3-1 Internal Structure of a B-tree Index



What kind of index is that: B/B^+ ? particularities?

B-trees in oracle: properties

Generally, index key associated to ROWID.

For composite indexes : optimizer may perform *index skip-scan*: searches one subindex per distinct value of first attribute (useful if low cardinality)

Index clustering factor : measures correlation between index order and repartition of records among data blocks.

Creating a B-tree index

```
CREATE INDEX cust_gender_adr_idx ON Customers(gender,address);
SELECT * FROM Customers WHERE address='36 Quai des Orfèvres';
```

B-trees: properties

Assets:

- ✓ Inserts, update, delete in $O(\log_m(n))$.
- ✓ Data type-independent (only requires order)¹
- ✓ Needs re-balancing after data modifications (not really an issue in DW)
- ✓ One-dimensional index structure: key = 1 attribute or sequence (not set).
- ✓ B^+ -tree good for range queries, on primary index.

Weaknesses:

- ✗ hardly adequate for multidimensional queries
- ✗ useful only when selectivity is high
(degenerate trees on low-cardinality attributes)

Ex: index on gender: 1.000.000 rows, $\approx 50\%$ M/F
 \Rightarrow 500.000 accesses. Table scan way faster.

¹can even be defined on a function of attributes

Index requirements for datawarehouses

- index may rely on ETL to enforce UNIQUE constraints.
 - queries often combine multiple conditions in WHERE clause
 - queries often join fact table with dimensions
-
- other indexes considerations: pre-aggregated levels, dimensions play symmetric role, cube data is sparse, batch updates should be efficient.
- ⇒ B-trees will be used almost exclusively for unique columns.

Typical ROLAP indexes:

- bitmap index
- bitmap join index

Bitmap Index

Idea: code value of attribute with bitmaps (bit-array).

Bitmap index on attribute a of relation R (with n records)

Data structure that stores one bitmap (n bits) per value of a :

$$i^{\text{th}} \text{ bit} = \begin{cases} 1 & \text{in the bitmap corresponding to value of } a \text{ in } i^{\text{th}} \text{ record} \\ 0 & \text{for other bitmaps} \end{cases}$$

Book

BookID	Title	Binding	Language
3240	Le Petit Prince	Hardcover	French
2211	Winnie the Pooh	Hardcover	English
9754	Paddington Bear	Paperback	NULL
4315	Pinocchio	Hardcover	Italian
2368	Les Vacances	Paperback	French

Bitmap index on *Binding*

Hardcover	Paperback
1	0
1	0
0	1
1	0
0	1

or rather:

Hardcover: 11010

Paperback: 00101

Bitmap Index

Idea: code value of attribute with bitmaps (bit-array).

Bitmap index on attribute a of relation R (with n records)

Data structure that stores one bitmap (n bits) per value of a :

$$i^{\text{th}} \text{ bit} = \begin{cases} 1 & \text{in the bitmap corresponding to value of } a \text{ in } i^{\text{th}} \text{ record} \\ 0 & \text{for other bitmaps} \end{cases}$$

Book

BookID	Title	Binding	Language
3240	Le Petit Prince	Hardcover	French
2211	Winnie the Pooh	Hardcover	English
9754	Paddington Bear	Paperback	English
4315	Pinocchio	Hardcover	Italian
2368	Les Vacances	Paperback	French

Bitmap index on Language?



Bitmap compression

Bit-vectors sparse on high-cardinality attribute:

- ✓ compression opportunities
 - ✗ compression may increase maintenance overhead
 - ✗ *must be decompressed at query time*
- ⚠ In OLTP: using compressed bitmap results in lock on many records.

Bitmap compression: RLE

Run-Length Encoding

Principle: *coding length of each block of repeated numbers: run-length*

Actually many variants of RLE. Here is one:

- a block (run) represents $k \geq 0$ "0" followed by a "1"
- block length written in binary
- add prefix to delimit each block (code must be unambiguous)
- trailing "0"s can be omitted (# records is known)

RLE: encodes a block of k "0" and a "1" ($\underline{k}_2 = k$ in binary):

- $k \geq 1$: $(|\underline{k}_2| - 1)$ "1", then "0", then \underline{k}_2 .
- $k = 0$: 00

Examples:

$\underline{3}_2$

000101 is encoded as 101101

100000000100001000 as: 00110111110100
 $\underbrace{100000000}_7$ $\underbrace{10000}_4$ $\underbrace{110100}_{\underline{7}_2}$ $\underbrace{110100}_{\underline{4}_2}$

Bitmap Indexes: benefits

Assets:

- ✓ no need to store ROWID: each bit-vector is much more compact than B-tree.
- ✓ because of small size, bitmaps usually fit into main memory.
- ✓ boolean operations on bitmaps very fast (AND, OR, XOR, NOT...)
- ✓ good on low-cardinality attributes
- ✓ not bad on high cardinality attributes:
- ✓ compression opportunities if sparse array.

Weaknesses:

- ✗ maintenance is costly: all indexes must be updated on insertion...
- ✗ storage space increases on high cardinality attributes
- ✗ decompression overhead (degenerate trees on low-cardinality attributes)

Bitmap indexes: queries with boolean operations

Book

BookID	Title	Binding	Language	Price
3240	Le Petit Prince	Stapled	French	35
2211	Winnie the Pooh	Hardcover	English	40
9754	Paddington Bear	Paperback	English	35
4315	Pinocchio	Stapled	Italian	15
2368	Les Vacances	Paperback	French	40

```
SELECT *
FROM Book
WHERE Binding != 'Stapled'
AND Price BETWEEN 20 AND 40;
```

Hardcover: 01000 40: 01001
Paperback: 00101 35: 10100
Stapled: 10010 15: 00010

NOT Stapled: 01101 35 OR 40: 11101

NOT Stapled AND (35 OR 40): 01101

Bitmap join index

Aim: index on relation R over attribute a in relation R' .

Precomputes the join(s) (spares joins at query time).

Stores one bitmap on R for each value of n' .

Sales

ShopID	BookID	Count
1	3240	2
1	2368	1
2	3240	4
2	9754	8
2	2211	5
3	9754	3

Book

BookID	Title	Binding	Language
3240	Le Petit Prince	Hardcover	French
2211	Winnie the Pooh	Hardcover	English
9754	Paddington Bear	Paperback	NULL
4315	Pinocchio	Hardcover	Italian
2368	Les Vacances	Paperback	French

Bitmap (join) index on
Sales over Binding:

Hardcover: 101010

Paperback: 010101

Bitmap/JOIN indexes in Oracle

Bitmap index

```
CREATE BITMAP INDEX binding_ix ON Book(binding);
```

Bitmap join index

```
CREATE BITMAP JOIN INDEX binding_bjix
ON Sales(Book.binding)
FROM Sales, Book
WHERE Sales.BookID=Book.BookID;
```



! JOIN INDEX ≠ INDEX JOIN

Star queries

Queries in DW are typically *star queries*, i.e., join the (huge) fact table to (small) dimension tables, but do not join any pair of dimensions.

(Traditional) DBMS joins are *pairwise*. Query optimizer then orders joins to minimize size of intermediate results. Traditional heuristic: only join relations connected through an attribute (no cross product).

Problem: only table related to others: Fact table.

Alternative approach: first compute cross product of (relevant) dimension rows, then join with fact table.

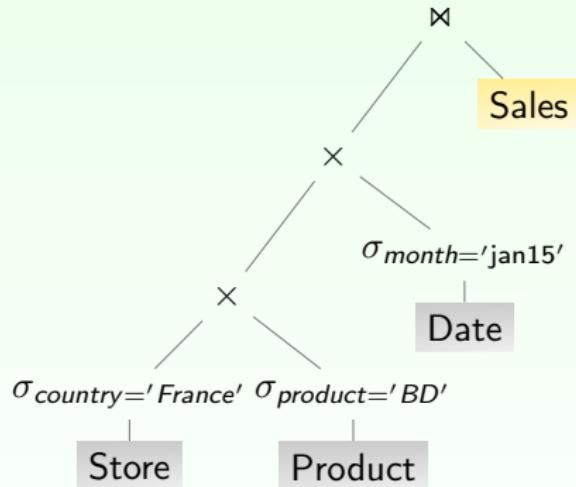
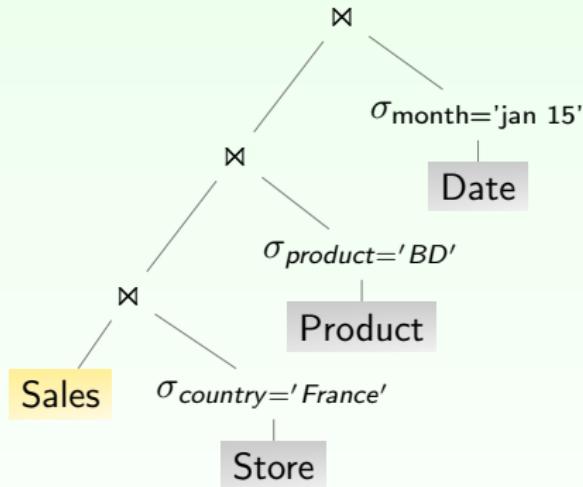
→ alternative may be more efficient when predicates are highly selective!

Bitmap indexes/ bitmap join indexes typically help optimize such queries.

Star queries (2)

Assume:

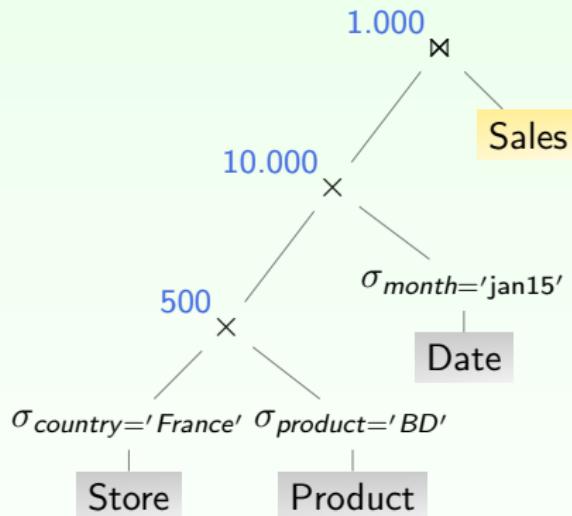
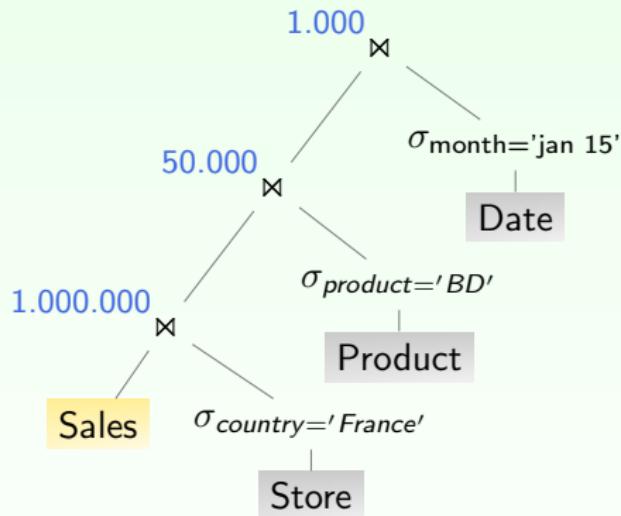
- 10.000.000 sales records
- query selects 10 stores (out of 100), 50 products (out of 1000), 20 days (out of 1000).
- uniform repartition of sales among stores, etc.



Star queries (2)

Assume:

- 10.000.000 sales records
- query selects 10 stores (out of 100), 50 products (out of 1000), 20 days (out of 1000).
- uniform repartition of sales among stores, etc.



Star queries (Oracle)

If there is a bitmap index (or bitmap join index) on all necessary foreign keys in the fact table, Oracle can then consider among the possible query execution plans what they call the *star transformation*:

- first select relevant tuples in the fact table
 - for this; select relevant IDs in each dimensions, then merge these IDs into a single bitmap per FK, then intersect the bitmaps
- then join the fact table to dimension tables



requires parameter STAR_TRANSFORMATION_ENABLED = true.

DB2 has similar approach, but uses one semijoin per dimension with B-tree on the FK instead of bitmaps.

Outline

M2 D&K 2017

⑥ Physical design and query optimization

- Partitioning
- Indexes
- Clustering

"Clustered indexes" in oracle

Usual tables are *heap-organized*.

Index-organized-table (Oracle): a B^+ -tree contains directly the data.
(index-only scan saves 1 indirection: the data block I/O)

Other indexes on table only store *logical* ROWIDs.

Creating an index-organized table

```
CREATE TABLE FrenchCities (
    zip int PRIMARY KEY,
    city_name VARCHAR2(20))
ORGANIZATION INDEX
MAPPING TABLE-- optional, creates a table mapping logical rowids
-- the mapping table is necessary to support bitmap indexes
;
```

"Clusters" in oracle: table clusters

When multiple tables share columns and are often queried together, they can be grouped in a *table cluster*:

- *Index cluster*: stores together (same block) rows from all tables with same value on cluster key.
- *Hash cluster*: stores together rows from all tables with same hash on cluster key.

Creating a clustered table

```
CREATE CLUSTER personnel(department NUMBER(4));
```

```
CREATE TABLE dept_10
CLUSTER personnel (department_id)
AS SELECT * FROM employees WHERE department_id = 10;
```

```
CREATE CLUSTER personnel(department NUMBER(4))
HASHKEYS 10;
--HASHKEYS should be approximately nb of unique values(performance)
```

--options may redefine hash function, storage size...

“Clusters” in oracle: table clusters

Assets:

- ✓ speed-up on joins (access-time, # of block I/O)
- ✓ saves space (key repetitions)

Weaknesses:

- ✗ maintenance cost
- ✗ table scan slower

Multidimensional clustering

Multicol. index weakness: 1) costly 2) useless when accessing non-prefix of key.

Multidimensional clustering on a table:

stores rows according to the value of cluster columns (physical proximity).

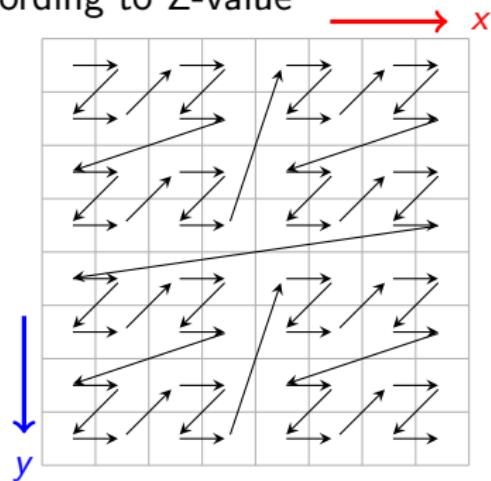
In oracle (≥ 12 only): supported through *attribute clustered* table. Store rows in an ordered way, where order can be:

- *linear ordering*: usual order on cluster column(s)
- *interleaved ordering*: data stored according to Z-value

rank of (i,j) in Z-curve:
interleave i_2 and j_2

$y \swarrow x \searrow$

ex: $(1,1) \rightarrow 11 = \underline{3}_2$
 $(2,0) \rightarrow 1000 = \underline{8}_2$



Creating attribute-clustered table in Oracle

(or reorganizing a table into a clustered one)

Creating an attribute-clustered table

```
CREATE TABLE sales (
    prod_id      NUMBER(6) NOT NULL,
    cust_id      NUMBER NOT NULL,
    time_id      DATE NOT NULL,
    amount_sold  NUMBER(10,2) NOT NULL
)
CLUSTERING BY INTERLEAVED ORDER (time_id, prod_id,cust_id);
-- CLUSTERING BY LINEAR ORDER (time_id, prod_id,cust_id);
```

Join attribute clustering

```
ALTER TABLE sales
ADD CLUSTERING
sales JOIN product ON (sales.prod_id=products.product_id)
BY INTERLEAVED ORDER (time_id, product_cat);
```

[<https://docs.oracle.com/database/121/DWHSG/attcluster.htm>]

“Clusters” in oracle: attribute clustering

Assets:

- ✓ no huge storage costs like indexes
- ✓ I/O reduction: allows direct access to relevant regions.
To achieve such savings in oracle; use in conjunction with
 - zone maps
 - in-memory min/max pruning
 - exadata storage indexes
 - interleaved ordering not affected by columns order
- ✓ improves compression ratio

Weaknesses:

- ✗ costly
- ✗ therefore oracle will not maintain the clustering after updates

“Clusters” in various DBMS

Some form of clustering supported in:

- MySQL(InnoDB):
clustered index
- Postgresql:
CLUSTER table: 1-time reordering, linear.
- DB2 (8,9...):
clustered index, MDC: creates one block index per column.
- SQL Server:
clustered index, columnstores for multidimensional clustering.

¹Z-curve also used by Oracle, SQL Server, DB2... for spatial objects.

References

Oracle Database Concepts

<https://docs.oracle.com/database/121/CNCPT/tablecls.htm>

Microsoft: clustered indexes and columnstores

<https://msdn.microsoft.com/en-us/library/ms189051>

<https://msdn.microsoft.com/fr-fr/library/gg492088>

DB2: indexes and MDC

https://www-01.ibm.com/support/knowledgecenter/SSEPGG_9.7.0/com.ibm.db2.luw.admin.dbobj.doc/doc/c0020180.html

https://www-01.ibm.com/support/knowledgecenter/SSEPGG_9.7.0/com.ibm.db2.luw.admin.partition.doc/doc/c0007201.html

<http://www.dbissoftware.com/db2nightshow/episode37slides.pdf>

[Padmanabhan and Cranston, SIGMOD'03]: available on Citeseer

PostgreSQL: Cluster

<http://www.postgresql.org/docs/9.1/static/sql-cluster.html>

(MultiDimensional) indexes: T. Grust's lecture

<http://db.inf.uni-tuebingen.de/teaching/DatenbanksystemeIISS2014.html>

Other physical storage structures

For multidimensional or spatial information:

- UB-tree: a B-tree on Z-order
- R-tree (and variants)
- kd-tree
- grid files
- multidimensional hashing

...

Other physical storage structures: Oracle's zone maps

Similar feature in IBM DB2.

Zone Map:

access structure that can be built on a table. Allows to prune blocks or partitions based on predicates during scans.

- stores min/max value of the relevant column(s) for each zone
- at most one per table, can include multiple columns
- implementation similar to materialized views
- can also store min/max value of another table through join

Properties:

- works best when data is clustered w.r.t. the zone map attributes
- more compact than indexes (granularity=1 zone instead of 1 row)
- the ranges for some zones may become stale

Interesting optimization technique: Bloom filters

Bloom filter:

probabilistic data structure to test membership in a set. Consists of:

- an array A of m bits
- k hash functions h_1, \dots, h_k

Array A for set S :

Initially $A = 0$. Then

$\forall x \in S \forall i \leq k$, set $A[h_i(x)]$ to 1.

To test if $y \in A$:

if $\bigwedge_i A[h_i(y)] = 0$ then $y \notin S$

if $\bigwedge_i A[h_i(y)] = 1$ then probably $y \in S$

Proba. of false positive:
 $p \approx (1 - e^{-kn/m})^k$.

Used:

- for partition pruning (ex: join and filter pruning in Oracle)
- for semi joins (DB2 star query optimization)
- to check if query result already in cache (Oracle)

Outline

M2 D&K 2017

7 Query optimization: Views

- Query rewriting
- View update
- Materialized Views: support by commercial RDBMS



Materialized views

Pre-computing and storing query results is a traditional approach to improve DB performance.

Materialized view

stores the tuples returned by a query

Materialized view can be queried like ordinary table, or can be used by optimizer for *query rewriting*. Typical usage:

- storing aggregates to reduce data volume for aggregate queries
- pre-compute "parts" common to multiple queries (joins, aggregates)

Assets:

✓ potentially drastic speedup at query-time

Weaknesses:

✗ storage space

✗ maintaining consistency with base table after updates.

Materialized views in DW

- Multiple similar queries (dimensional schema)
 - difference with OLTP: a MV is for the query optimizer; OLTP uses non-materialized views mostly, to simplify queries/data presentation.
 - query rewriting gets simpler: common structure, queries repetitive
- Mostly read-only queries; updates not frequent
 - infrequent updates : maintenance effort is moderate
 - speedup on recurring 'hard' queries

→ *materialized views have been popular tool in DW.*

Materialized views: main issues

- Make the most of materialized views when evaluating queries.
 - find the best query plan in presence of materialized views
 - generally a hard problem
- Keep the view consistent when base table is updated
 - recompute from scratch
 - or incremental view maintenance
- Choosing the data to materialize
 - depends on storage space available and on query patterns
 - can be hard too.

Outline

M2 D&K 2017

7 Query optimization: Views

- Query rewriting
- View update
- Materialized Views: support by commercial RDBMS

Query rewriting: principle

Query equivalence

Queries Q and Q' on relational schema R are *equivalent* if they return the same result on all possible instances:

$$Q \equiv Q' \quad \text{iff} \quad \forall D, Q(D) = Q'(D)$$

Query rewriting

Given a view (or set of views) V and query Q on schema R ,
Queries Q' is a *rewriting of Q using V* if its schema is $R \cup V$ and $Q' \equiv Q$.

A query rewriting may also use multiple views.

Ideally, rewriting uses view(s) only: $\forall D, Q''(V(D)) = Q(D)$.

Query rewriting: example

filtering

Materialized view: sales 2015

```
-- Sales15_mv:  
SELECT S.ID sid, P.name pname  
FROM Sales S, Product P  
WHERE S.PID=P.PID  
AND S.year='2015'
```

Query Q: chair sales in 2015

```
SELECT S.ID id  
FROM Sales S, Product P  
WHERE S.PID=P.PID  
AND P.name='chairs'  
AND S.year='2015'
```

$Q \equiv$

```
SELECT sid id  
FROM Sales15_mv  
WHERE pname='chairs'
```

filter on the view instead of all sales: rewritten query is much faster.

Query rewriting: example

aggregates

Materialized view: sales 2015

```
-- Sales_mv:  
SELECT country, day, SUM(Amount)  
FROM Sales S  
    NATURAL JOIN Customer C  
    NATURAL JOIN Time T  
GROUP BY country,day
```

Query Q: chair sales in 2015

```
SELECT month, SUM(Amount)  
FROM Sales S  
    NATURAL JOIN Customer  
    NATURAL JOIN Time  
WHERE country='FRANCE'  
GROUP BY month
```

$Q \equiv$

```
SELECT month, SUM(Amount)  
FROM Sales_mv  
    NATURAL JOIN Time  
WHERE country='FRANCE'  
GROUP BY month
```

$|Sales_mv| \ll |Sales|$: rewritten query is much faster.

Output size estimation

Pipesort and many query optimization algos need output size estimates.

- Expected # of distinct tuples when draw r among n possible values *uniformly*:

$$n - n(1 - 1/n)^r$$

↪ no data access, but strong independance assumption (independance, uniform).

- Estimating unknown # of distinct tuples k in a multiset M :

hash-based algorithm estimates k with single scan of M (constant time per item) and logarithmic space.

[Kane et al, PODS'10], based on [Flajolet and Martin FOCS'83]

Both can be used to estimate answer cardinality for a GROUP BY query (M= original table, r=number of lines in M).

Query rewriting in Oracle (1)

```
CREATE MATERIALIZED VIEW product_sales_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_name, SUM(s.amount_sold) AS dollar_sales
FROM sales s, products p WHERE s.prod_id = p.prod_id
GROUP BY p.prod_name;
```

For query rewriting to be considered:

- materialized views considered if **ENABLE QUERY REWRITE** clause
- session parameter **QUERY_REWRITE_ENABLED** must be set to **TRUE** (default) or **FORCE** (adopts rewriting even if cost without rewrite is lower).
- must use cost-based optimization with **OPTIMIZER_MODE** among
 - **ALL_ROWS**
 - **FIRST_ROWS** (optimizes delay for first rows)
 - **FIRST_ROWS_n**
- if some constraints are not validated, you may want to modify session parameter **QUERY_REWRITE_INTEGRITY**:
 - **ENFORCED (default)** : exploits fresh data and **ENABLE VALIDATED** constraints
 - **TRUSTED** : uses declared constraints even if not validated
 - **STALE_TOLERATED** : uses views that contain stale data

Outline

M2 D&K 2017

7 Query optimization: Views

- Query rewriting
- **View update**
- Materialized Views: support by commercial RDBMS



Materialized views

Update

Materialized views store result of query on base tables. When base tables are updated (ETL), MV must be updated!

Two options:

- Re-compute view from scratch

delete all view records, re-evaluate the view query on database. Often too costly

- Incremental maintenance

compute new state of the view from old one and list of modifications.

$$V' = V - \Delta_V^- \cup \Delta_V^+$$

Incremental maintenance generally more efficient, but:

- generally still need to query part of the base tables
- can be more costly than re-materialization if larger number of updates

Materialized views in oracle: maintenance (1)

```
CREATE MATERIALIZED VIEW product_sales_mv
BUILD DEFERRED
REFRESH COMPLETE|ON DEMAND
ENABLE QUERY REWRITE AS
SELECT p.prod_name, SUM(s.amount_sold) AS dollar_sales
FROM sales s, products p WHERE s.prod_id = p.prod_id
GROUP BY p.prod_name;
```

BUILD DEFERRED/IMMEDIATE: view remains empty until first update
/populated immediately

ON DEMAND/ON COMMIT: update when requested/after each transaction

COMPLETE : recomputes view from base tables

FAST : tries to update incrementally using logs or PCT

FORCE : tries to update incrementally, otherwise complete

Materialized views in oracle: maintenance (2)

New maintenance option in Oracle 12: SYNCHRONOUS REFRESH, which updates base table and view at the same time.

Several steps:

- register table (create view log) and view (statement) for synchronous refresh

after this one-time registration: table cannot be updated except as follows:

- prepare the change to data in outside tables
- execute the refresh operation (the system performs partition exchange operations)

```
-- registration:  
CREATE MATERIALIZED VIEW LOG ON fact FOR SYNCHRONOUS REFRESH USING st_fact;  
EXECUTE DBMS_SYNC_REFRESH.REGISTER_MVIEWS('MV1');  
-- refresh operations  
EXECUTE DBMS_SYNC_REFRESH.PREPARE_REFRESH(DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));  
EXECUTE DBMS_SYNC_REFRESH.EXECUTE_REFRESH(DBMS_SYNC_REFRESH.GET_GROUP_ID('MV1'));
```

Conditions on parameters update ON DEMAND, integrity TRUSTED.

Materialized views maintenance in oracle: logs (1)

Logs are necessary for incremental maintenance (except for PCT refresh). Can be conventional (fast refresh) or staging (synchronous refresh) logs.

- Can be based on SCN (only for conventional), or timestamps (default).
- Should include ROWID for joins, SEQUENCE for mixed insert/delete/update and INCLUDING NEW VALUES for aggregates.

```
CREATE MATERIALIZED VIEW LOG ON sales WITH ROWID;
CREATE MATERIALIZED VIEW LOG ON times WITH ROWID;
CREATE MATERIALIZED VIEW LOG ON customers WITH ROWID;
```

```
CREATE MATERIALIZED VIEW detail_sales_mv
BUILD IMMEDIATE
REFRESH FAST AS
SELECT s.rowid "sales_rid", t.rowid"times_rid", c.rowid "customers_rid",
c.cust_id, c.cust_last_name, s.amount_sold, s.quantity_sold, s.time_id
FROM sales s, times t, customers c
WHERE s.cust_id = c.cust_id(+) AND s.time_id = t.time_id(+);
```

[<https://docs.oracle.com/database/121/DWHSG/basicmv.htm>]

Materialized views maintenance in oracle: logs (2)

```
CREATE MATERIALIZED VIEW LOG ON products WITH SEQUENCE, ROWID  
(prod_id, prod_name, prod_desc, prod_subcategory, prod_subcategory_desc,  
prod_category, prod_category_desc, prod_weight_class, prod_unit_of_measure,  
prod_pack_size, supplier_id, prod_status, prod_list_price, prod_min_price)  
COMMIT SCN INCLUDING NEW VALUES;  
  
CREATE MATERIALIZED VIEW LOG ON sales  
WITH SEQUENCE, ROWID  
(prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold, amount_sold)  
COMMIT SCN INCLUDING NEW VALUES;  
  
CREATE MATERIALIZED VIEW product_sales_mv  
PCTFREE 0 TABLESPACE demo  
STORAGE (INITIAL 8M)  
BUILD IMMEDIATE  
REFRESH FAST  
ENABLE QUERY REWRITE  
AS SELECT p.prod_name, SUM(s.amount_sold) AS dollar_sales,  
COUNT(*) AS cnt, COUNT(s.amount_sold) AS cnt_amt  
FROM sales s, products p  
WHERE s.prod_id = p.prod_id GROUP BY p.prod_name;
```

References

Oracle Data Warehousing Guide

<https://docs.oracle.com/database/121/DWHSG/toc.htm>

View logs in Oracle:

https://docs.oracle.com/database/121/SQLRF/statements_6003.htm

Other infos about MV in Oracle:

<https://docs.oracle.com/database/121/REPLN/repview.htm>

Outline

M2 D&K 2017

7 Query optimization: Views

- Query rewriting
- View update
- Materialized Views: support by commercial RDBMS



Materialized views support in some DBMS

Materialized views can be simulated through CTAS and triggers.

Oracle and IBM DB2 provide support for the features discussed.

Other DBMS generally provide partial/indirect support.

In IBM DB2: called *materialized query tables*.

IBM DB2

```
CREATE TABLE Sales.vOrders [(column [ ,...n ])]
AS select_statement
[DATA INITIALLY DEFERRED]
REFRESH [DEFERRED / IMMEDIATE]
```

DB2 supports query rewriting, incremental refresh, etc.

Materialized views support in some DBMS

Materialized views not directly supported in SQL Server but INDEXED VIEWS are the closest equivalent:

An indexed view is always kept up-to-date (refresh *on commit*).

Query optimizer considers using the view for query rewriting in Enterprise or Developer edition, but not others.



SQL Server

```
CREATE VIEW Sales.vOrders [(column [ ,...n ])]  
WITH SCHEMABINDING
```

```
AS select_statement
```

```
CREATE UNIQUE CLUSTERED INDEX index_name  
ON Sales.vOrders [(column [ ,...n ])]
```

Materialized views support in some DBMS



Postgresql

```
CREATE MATERIALIZED VIEW view_name  
[(column_name [ ,... ])]  
AS select_statement  
[WITH [NO] DATA]
```

Can be refreshed, only support *full, on demand* refresh.

No direct support in Mysql, MariaDB.



But possible support through SQL API flexviews
(refresh *on demand* only, can be *full* or *incremental*).

Outline

M2 D&K 2017

8 Building the DW

- Architecture
- Data integration: the ETL process

Outline

M2 D&K 2017

8 Building the DW

- Architecture
 - Requirements and reference architecture
 - Metadata
- Data integration: the ETL process

Architecture

architecture: process and product of designing structures

Functional requirements for DW architectures:

- support for multiple analysis tools
- independence of analyses from sources (updates, availability)
- customized views
- adaptability
- scalability
- extensibility (new sources, analytical tools)
- integration and homogenization of multiple data sources
- historization of data
- quality up-to-date data
- cost efficiency
- automated processes

12 rules for OLAP [Codd et al.,93]

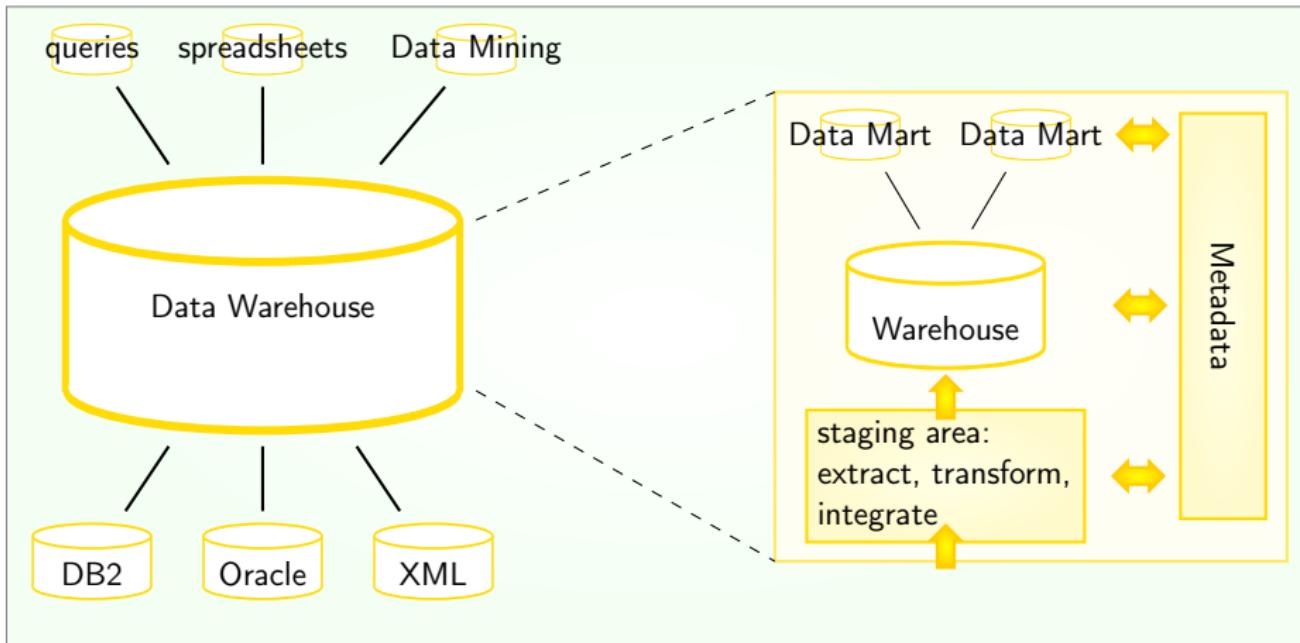
Multidimensional conceptual view	Supports slice and dice operations...
Transparency	Supports heterogeneous data sources. The end user should not be concerned about the details of data access or conversions
Accessibility	Presents the user with a single logical schema of the data.
Reporting performance	Performance should not degrade much as dimension number increases
Client/server architecture	Server must allow to attach new clients with minimum effort and programming for integration
Generic dimensionality	Dimensions should have similar structure and operational capabilities
Dynamic sparse-matrix handling	Accommodates varying storage and data-handling options
Multiuser support	Supports multiple concurrent users/views on a common database
Unltd. Crossdimensional operations	All dimensions are created equal, so all forms of calculation must be allowed across all dimensions, not just the measures dimension
Intuitive data manipulation	Users shouldn't have to use menus or perform complex multiple step operations for simple operations
Flexible reporting	Users should be able to print just what they need, and any changes to the underlying model should be automatically reflected in reports
Unltd. dimension &aggregation levels	Essentially no limit on the number of dimensions supported and on the number of levels per dimension

FASMI rules [Pendse'05]

Fast Analysis of Shared Multidimensional Information (FASMI)

- fast answers
- simple and flexible analysis
- heterogeneous users with different access rights
- multidimensional data model
- queries filtering dimensions, with range values on attributes

Bird's view on architecture



Birds' view on DW architecture

Metadata

metadata: data that describes data

Aims:

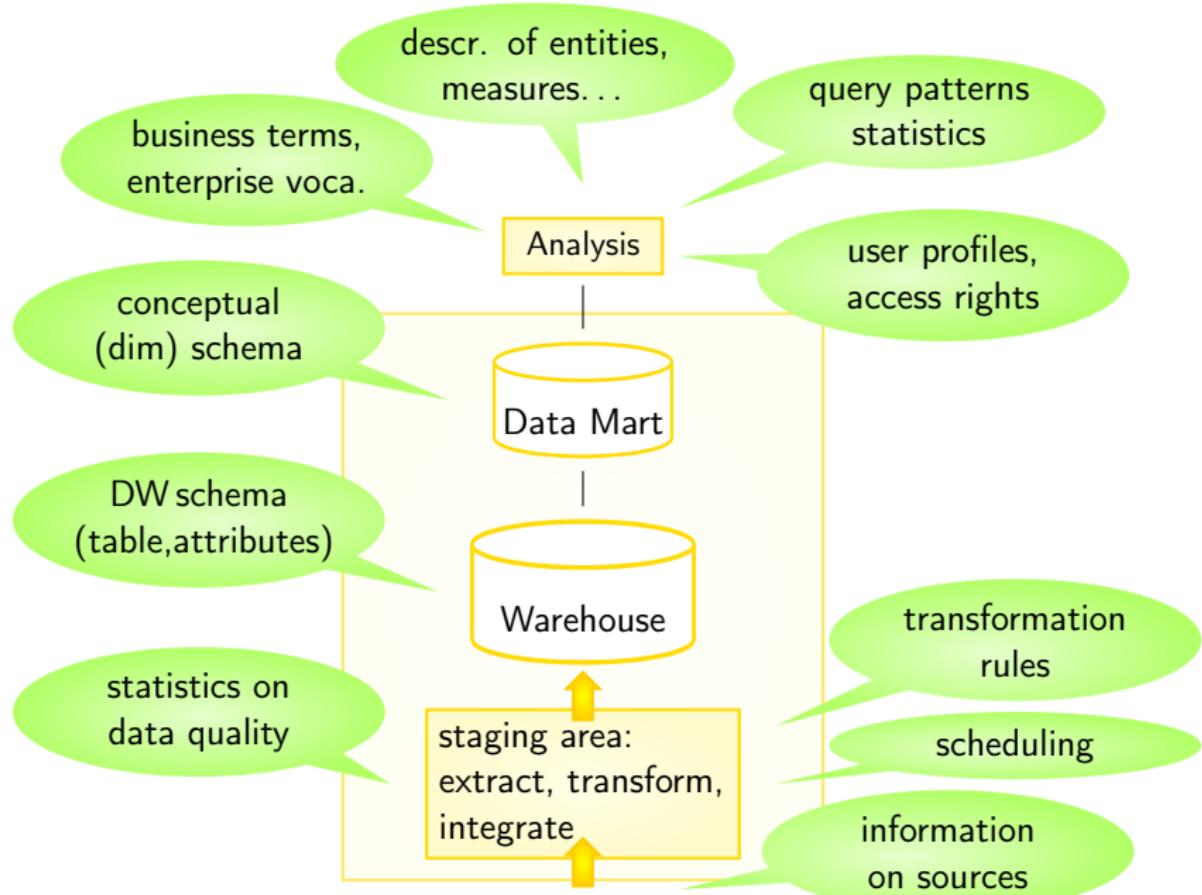
- monitor and understand processes
- avoid erroneous interpretations
- describe technical aspects of the DW

Multiple proprietary models, some standardization effort: *Common Warehouse Metamodel* from OMG.
(other standards: IRDS, OIM...)

	Warehouse Process			Warehouse Operation			
	Transformation		OLAP	Data Mining	Information Visualization	Business Nomenclature	
Management	Object Model		Relational	Record	Multidimensional		XML
	Business Information	Data Types	Expression	Keys and Indexes	Type Mapping	Software Deployment	
Object Model							

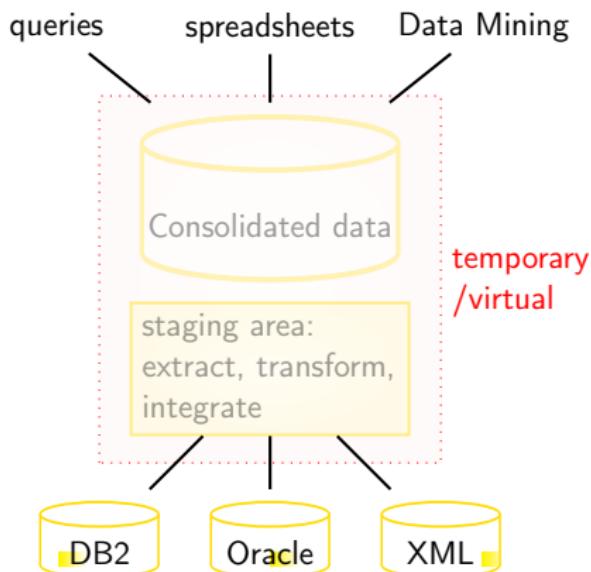
The CWM metamodel

Metadata



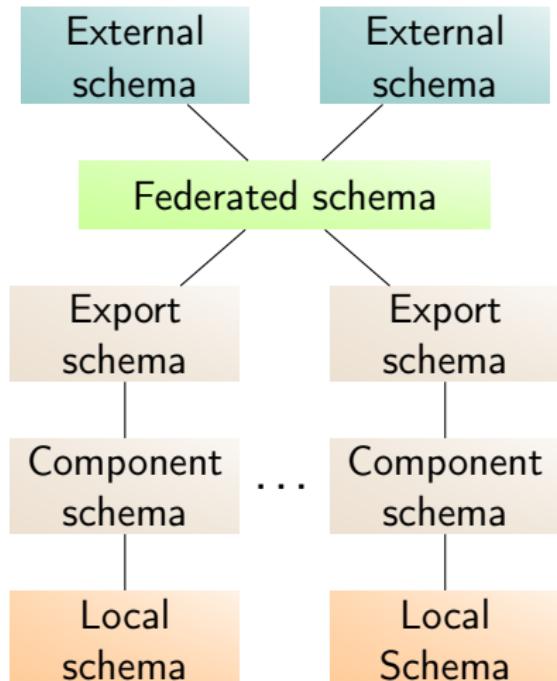
Materialized vs virtual data

- Data marts may be virtual or materialized
- Metadata persistent
- Staging area typically temporary
- DW typically persistent except in virtual integration:



- data stored in sources only
- only data relevant to query is extracted, and transformed on-the-fly.
- global schema allows query formulation
- no persistent storage \Rightarrow no history
- architecture may be:
 - 5 levels federated database
 - mediator wrapper

5 levels federated DB



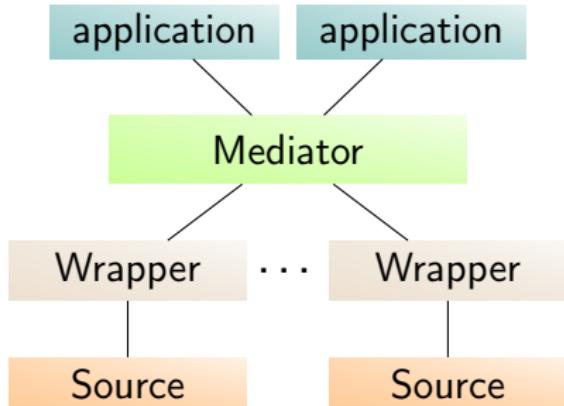
Solution to provide common access to autonomous sources.

- logical schema of source is mapped into component schema (solves heterogeneity issues)
- Export schema is a portion of component schema (access control...)
- Federated schema integrates export schemas, is aware of data repartition. Might be large.
- External schema implements access control, simplifies federated schema, hides schema evolutions.

Metadata helps!

Federated: autonomous DB sharing their data (pairwise) (minimal centralization)

Mediator-Wrapper



Solution to provide common access to heterogeneous independent sources (not relying on metadata).

- wrapper is buffer level between sources and mediator; solves heterogeneity issues (interfaces, data model, semantics)
- Mediator integrates the data *but mediator schema derived from applications and not from source integration.*

Remote data sources in DBMS

SAP Hana (SDA)

```
-- create remote access
CREATE REMOTE SOURCE HIVE1 ADAPTER "hiveodbc"
CONFIGURATION 'DSN=hive1'
WITH CREDENTIAL TYPE 'PASSWORD' USING
'user=dfuser;password=dfpass';

-- wrap remote source as virtual table
CREATE VIRTUAL TABLE "VIRTUAL_PRODUCT"
AT "HIVE1"."dflo"."dflo"."product";

-- query the data
SELECT product_name, brand_name
FROM "VIRTUAL_PRODUCT";
```

[<https://openproceedings.org/2015/conf/edbt/paper-339.pdf>]

Part of query execution plan can be forwarded to remote source.
Choices may impact performance.

Remote data sources in DBMS (2)



PostgreSQL

```
CREATE EXTENSION postgres_fdw;

-- create remote access
CREATE SERVER hive1
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'foo', dbname 'foodb', port '5432');

-- -- authentication credentials
CREATE USER MAPPING FOR CURRENT_USER
    SERVER hive1
    OPTIONS (user 'jdoe', password 'secret1');

-- import distant schema
CREATE SCHEMA app;

IMPORT FOREIGN SCHEMA public -- or CREATE FOREIGN TABLE up (...) SERVER hive1
    FROM SERVER hive1
    INTO app;

-- query the data
SELECT product_name, brand_name
FROM app1.VIRTUAL_PRODUCT;
```

Outline

M2 D&K 2017

8 Building the DW

- Architecture
- Data integration: the ETL process
 - Load



Data staging: the ETL process

Objective: integrate most up-to-date data in DW.

Traditionally, data physically transported in two steps:

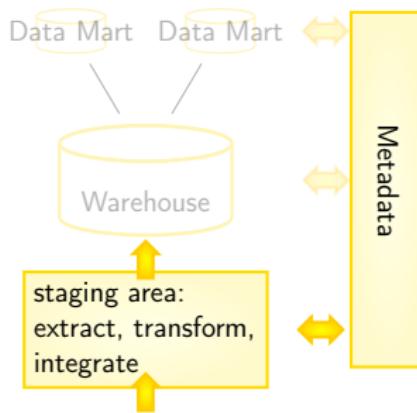
- *from sources to staging area:*

extract the data, differential updates ⇒ minimize load on OLTP.

- *from staging area to base database:*

cleaning, filtering and integration ⇒ achieve data quality (consistency...)

ETL considered most demanding in DW development (up to 80% of effort)!



ETL

ETL

- *Extraction*: selecting relevant data from sources
- *Transformation*: adapting data to the target schema, meet quality requirements
- *Loading*: feeding the data from staging area into target database

Challenges for ETL:

- multiplicity of sources
- heterogeneity
- volume
- complex transformations
- recovery from failure

2 major alternatives for extraction:

- **snapshot extraction** (whole data)
- **incremental extraction** (requires source cooperation)

Extraction tools on DBMS

Legacy/Non DB:

- on the host system without monitoring: batch jobs, reportwriter,
- on non-standard DBs: programs (cobol, pl-1,IMS...)

DBMS:

- replication-based: DBMS replicates changes in separate DB (ex: Oracle Streams, DB2DataPropagator)
- log analysis (ex: Oracle's LogMiner): deduce changes from transaction logs
- snapshots (for small dataset)
- triggers
- timestamps (may exploit range partitioning)

Load

Loading data into DW

Critical issue is efficiency: writing may lock the DW's fact table.

Performance improves when:

- separate update from inserts
- utilize bulk load: DBMS can insert batch of rows very fast
 - no logs
 - disable integrity constraints
 - disable triggers
 - lock whole table
 - better to drop indexes and rebuild afterwards
 - exploit parallelism
 - exploit partition exchange

Need failure recovery!

Remark: dimension table records need not necessarily be deleted when the event referring them are (apply business rules).

Loading: Merge

Separate update from inserts?

When loading a record, operations typically depend on whether value refers to existing DW record or must create new one.

```
MERGE INTO customer C          --table modified
    USING (SELECT * FROM new_cust) N  --source data
    ON (C.name = N.name)            --join condition
    WHEN MATCHED THEN UPDATE SET ...
    WHEN NOT MATCHED THEN INSERT VALUES (...);
```

⇒ more efficient than traditional 2-step technique

SQL 2003 instruction (sometimes called UPSERT).

Supported by Oracle, DB2, SQL Server.

MySQL and MariaDB: INSERT ... ON DUPLICATE KEY UPDATE (restriction on join condition: Unique key).

PostgreSQL: INSERT ... ON CONFLICT(...) DO.

Loading: Oracle's sql loader

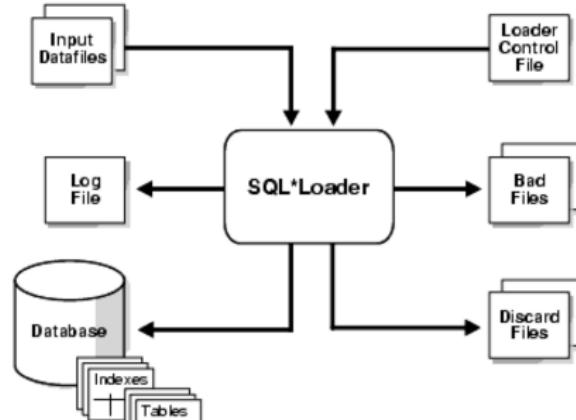
Control file:

```
LOAD DATA
  INFILE 'c:\data\mydata.csv'
  INTO TABLE emp
  fields terminated by "," optionally enclosed by ''
  ( empno, empname, sal, deptno )
```

Data file:

```
10001,"Scott Tiger", 1000, 40
10002,"Frank Naude", 500, 20
```

[http://www.orafaq.com/wiki/SQL*Loader_FAQ]



Other tools/instructions relevant for ETL:

- oracle data pump
- INSERT /*+ APPEND */
- multitable inserts...

```
INSERT ALL
  WHEN (q1+q2+q3+q4 <= 100) THEN
    INTO LowSALES VALUES (dealer,q1+q2+q3+q4)
  WHEN (q1+q2+q3+q4 > 100) THEN
    INTO HighSALES VALUES (dealer,q1+q2+q3+q4)
  SELECT * FROM sales
```

ETL vs ELT

Extract, Load, Transform (ELT) performs transformation in the target database, typically using SQL (ex: CTAS).

- Recent trend with cloud engines and clusters, to exploit computing power of target system.
- Might or might not perform as well as mature tools for pipelined ETL, but does not require a transformation engine (cost saving, simplifies development).

ETL dedicated tools (history?)



Commercial:

- Oracle Data Integrator
- Microsoft SSIS
- IBM InfoSphere DataStage
- Informatica
- ...



Open source:

- Talend
- Pentaho data integration (previously kettle)
- CloverETL
- ...



Pentaho Data Integration
Previously kettle

References

- *The Data Warehouse ETL Toolkit*, Kimball, Caserta.
- *Dimensional Modeling: In a Business Intelligence Environment*, IBM redbooks
- <http://www.oracle.com/us/products/middleware/data-integration/di-oracle-informatica-ibm-wp-2438402.pdf>

Data quality:

- *Information Quality Applied: Best Practices for Improving Business Information, Processes and Systems*, Larry English
- *Data Quality: The Accuracy Dimension*, Jack Olson
- https://it.ojp.gov/documents/informatica_whitepaper_monitoring_dq_using_metrics.pdf

Outline

M2 D&K 2017

9 In-memory column stores

- Introduction to in-memory databases (insp. by H.Plattner's lectures)
- In-memory databases: commercial systems: MariaDB
- In-memory databases: commercial systems: Oracle
- In-memory databases: commercial systems: IBM DB2
- Techniques for handling updates

Outline

M2 D&K 2017

9 In-memory column stores

- Introduction to in-memory databases (insp. by H.Plattner's lectures)
- In-memory databases: commercial systems: MariaDB
- In-memory databases: commercial systems: Oracle
- In-memory databases: commercial systems: IBM DB2
- Techniques for handling updates

Hardware: data storage

The hard drive

Created in 1956 (IBM).

Multiple disks (platters) spinning rapidly together (thousands rpm/min).

Ferromagnetic. Multiple tracks per platter. 1 head per platter, on air cushion.

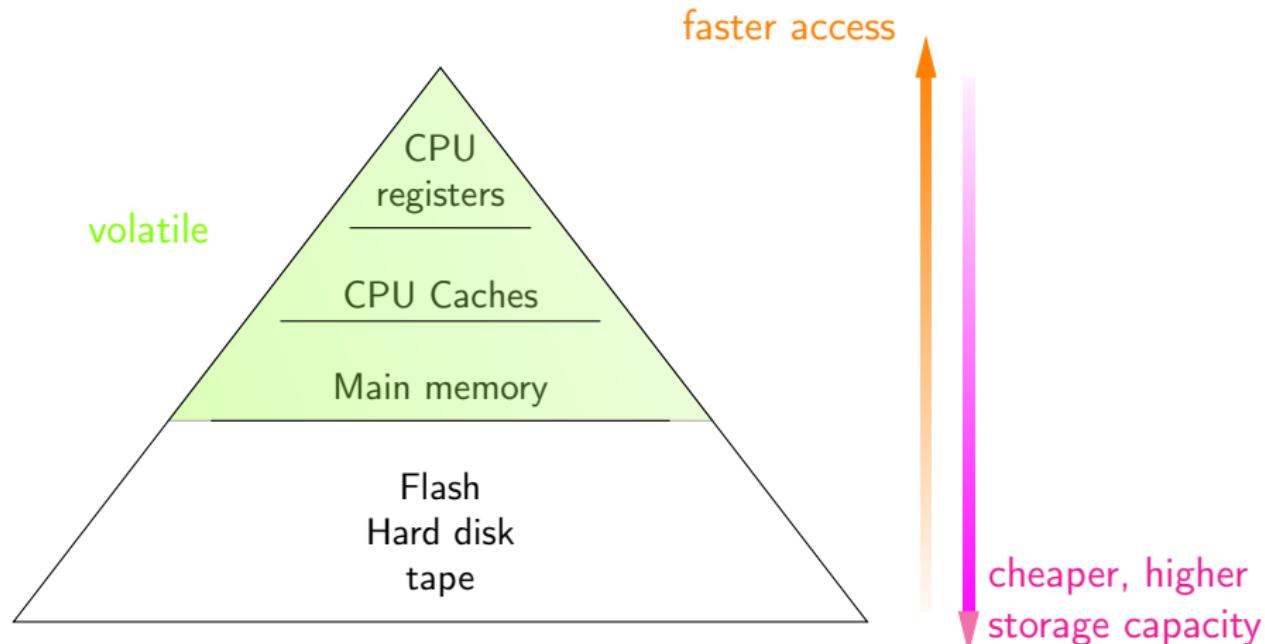


≈ 100€
a few TB
100gr.

Main storage device (secondary storage) since 1960.
The support for which DBMS were devised.

Hardware: storage

Memory hierarchy:



On early computers, CPU frequency \approx memory bus and memory access.
But CPU became much faster than memory.



New DB technologies and trends

Data storage and processing evolutions:

- New hardware (FPGA, GPGPU, SSD, NVRAM)
- In-memory DB/ Column storage
- NoSQL: (shared nothing) massively parallel architecture
(Map/Reduce, key/value)

Trends emphasize

- supporting both *analytical* (OLAP, mining) and *prediction* queries (ML, esp. *deep learning*)
- distributed data processing (“Hadoop” stack)
- trust management – so far, more of an issue for transactions rather than storage/analysis (blockchain)
- integrating new (external) types of data (text, feeds, images, video...)
- building data summaries (data too large to be stored): sampling, etc.
- real-time DW

Column-oriented DB = column store = column-major

Column storage vs Row storage

Idea: modify traditional storage for relation R . In consecutive memory, instead of storing a row we store a column (values of one attribute $R.x$).

Assets:

- ✓ only accesses relevant attributes
- ✓ potentially drastic speedup at query-time, esp. aggregation
- ✓ better compression techniques (values from same domain: many identical)
- ✓ allows vectorization (bitwise, SIMD)

Weaknesses:

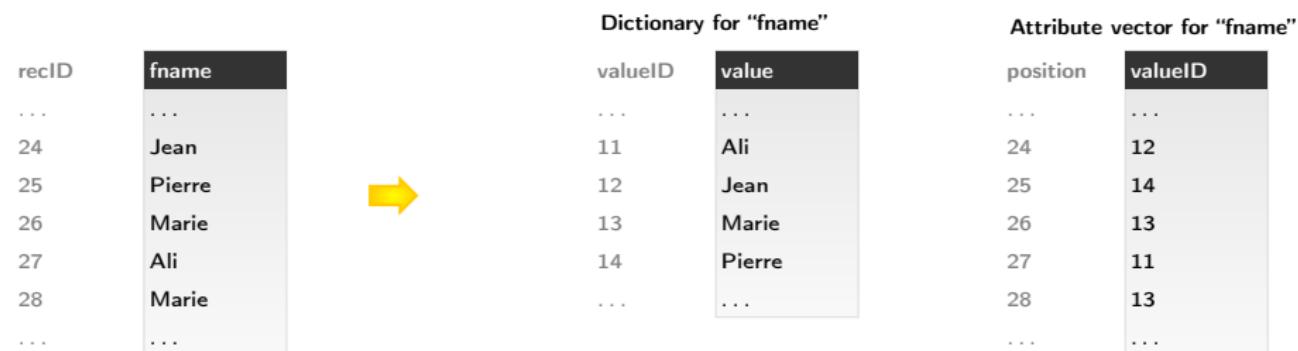
- ✗ need fast tuple reconstruction
 - ✗ slower on `select *`
 - ✗ updates (insertions, deletions...) are harder
- ⇒ analytical workloads, mostly reads, large data

Column-oriented DB

Dictionary encoding

 *The following slides describe the Sanssouci prototype architecture. Similar ideas (dictionary, buffering updates) apply in other column-oriented DB but with significant variations.*

Objective: reduce main memory operations through (lightweight) compression.



[Plattner, in-memory databases course]

The dictionary is sorted \implies fast lookup of id from value, fast range queries.

Column-oriented DB

Compression

valueID is already a “compressed” representation of value. But we can compress attribute vector (e.g.: RLE).

Column	Cardinality	Bits Needed	Item Size	Plain Size	Size with Dictionary (Dictionary + Column)	Compression Factor
First names	5 millions	23 bit	50 Byte	400GB	250MB + 23GB	≈ 17
Last names	8 millions	23 bit	50 Byte	400GB	400MB + 23GB	≈ 17
Gender	2	1 bit	1 Byte	8GB	2b + 1GB	≈ 8
City	1 million	20 bit	50 Byte	400GB	50MB + 20GB	≈ 20
Country	200	8 bit	47 Byte	376GB	9.4kB + 8GB	≈ 47
Birthday	40000	16 bit	2 Byte	16GB	80kB + 16GB	≈ 1
Totals			200 Byte	≈ 1.6TB	≈ 92GB	≈ 17

[Plattner, in-memory databases course]

Dealing with updates

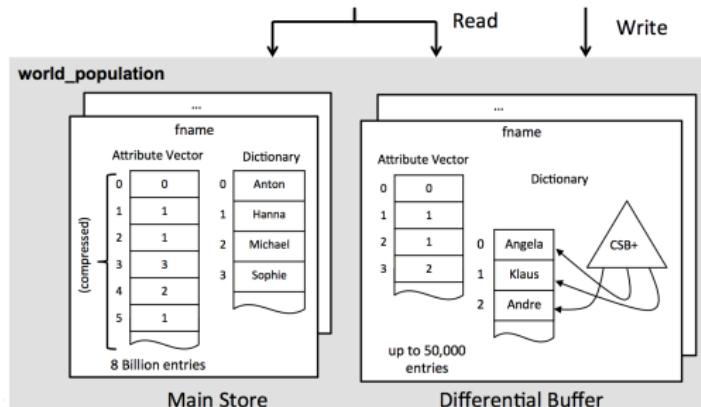
The differential buffer

Inserting a value may:

- have no impact on dictionary
- add a value at the end of dictionary (#bits may change)
- force a dictionary reorganization (sorted dict)

→ may reorganize the whole attribute vector (same for deletions).

⇒ we keep the main store *read-only*. Perform insert, update, delete on the differential buffer only.



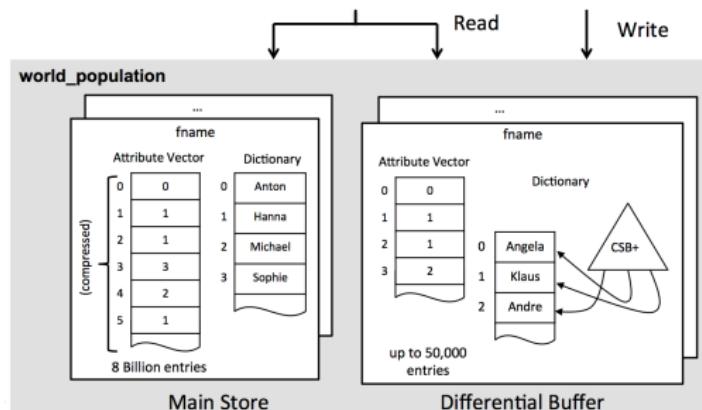
The differential buffer

Queries will check both the *compressed main store* and *differential buffer*.

The differential buffer:

- records updates
- is kept small (periodically merged into the main store and emptied)
- uses column storage but with *unsorted* dictionary
- an index (CSB+-tree) is maintained on the dictionary

A validity attribute is added to tuples (uncompressed bit vector in main store)...



The differential buffer: updates

Validity attribute: uncompressed bit vector in main store.

Dealing with updates/deletions:

- we update validity attribute in main store
- insert corresponding tuple in differential buffer

recid	fname	Iname	gender	country	city	birthday	valid
0	Martin	Albrecht	m	GER	Berlin	08-05-1955	1
1	Michael	Berg	m	GER	Berlin	03-05-1970	0
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968	1
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992	1
4	Ulrike	Schulze	f	GER	Potsdam	09-03-1977	1
5	Sophie	Schulze	f	GER	Rostock	06-20-2012	1
...
8×10^9	Zacharias	Perdopolus	m	GRE	Athen	03-12-1979	1

Main Store

Michael Berg moves
to Potsdam

0	Michael	Berg	m	GER	Potsdam	03-05-1970	1

Differential Buffer

[Plattner, in-memory databases course]

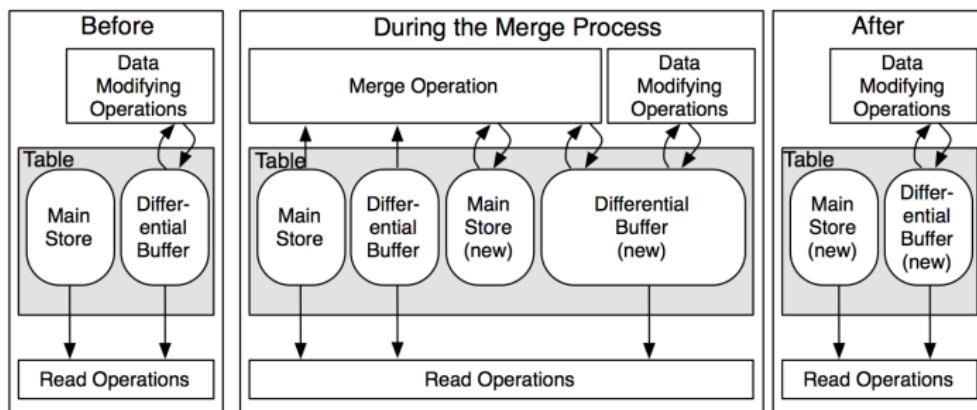
One possible way to deal with deletions: do not delete: keep validity interval (like scd type 2) \Rightarrow insert-only approach.

Merging the differential buffer and main store: architecture

Data in main store takes less space (compression) and benefits from faster reads (sorted) \Rightarrow keep differential buffer small \Rightarrow merge process.

We first create a second (empty) differential buffer: updates during (and after) the merge are directed toward that new buffer.

- ✓ Advantage of working on copies: short lock.
- ✗ Drawback: needs dedicated resources (2x space).



[Plattner, in-memory databases course]

Merge process: 1) combine dictionaries 2) compute new attribute vector.

The Merge process

Merge process: 1) combine dictionaries 2) compute new attribute vector.

Main Store

Dictionaries		Attribute vectors		Validity vector			
valueID	fname	city	recID	fname	city	recID	valid
0	Albert	Berlin	0	2	0	0	0
1	Michael	London	1	1	1	1	0
2	Nadja		2	0	0	2	1

Differential Buffer

Dictionaries		Attribute vectors		Validity vector			
valueID	fname	city	recID	fname	city	recID	valid
0	Michael	Berlin	0	0	0	0	0
1	Nadja	Potsdam	1	1	1	1	1
2	Hanna	Dresden	2	0	1	2	1
			3	2	2	3	1

The Merge process (2)

Merge process: 1) combine dictionaries 2) compute new attribute vector.

Main Store (new)

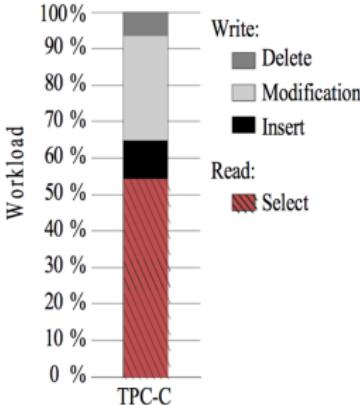
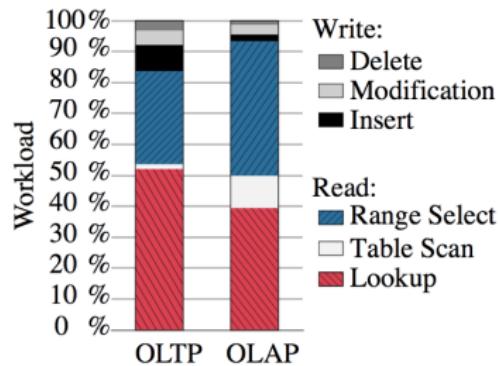
Dictionaries			Attribute vectors			Validity vector	
valueID	fname	city	recID	fname	city	recID	valid
0	Albert	Berlin	0	3	0	0	0
1	Hanna	Dresden	1	2	2	1	0
2	Michael	London	2	0	0	2	1
3	Nadja	Potsdam	3	2	0	3	0
			4	3	3	4	1
			5	2	3	5	1
			6	1	1	6	1

Differential Buffer (new)

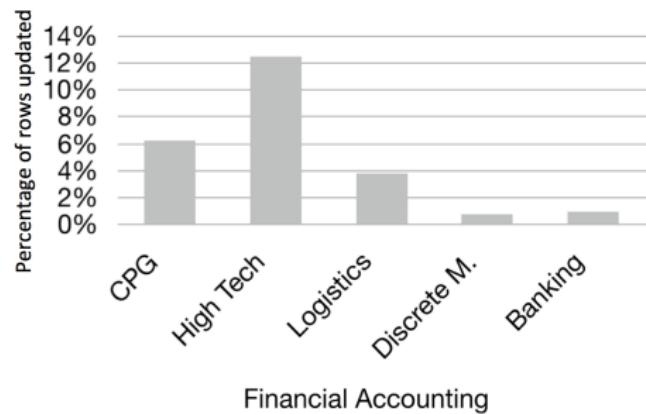
Dictionaries			Attribute vectors			Validity vector	
valueID	fname	city	recID	fname	city	recID	valid

Column-oriented DB

OLTP vs OLAP



Few updates in OLTP



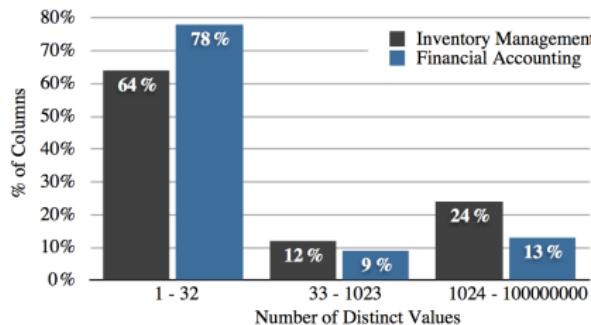
[Plattner, in-memory databases course]

Column-oriented DB

OLTP vs OLAP

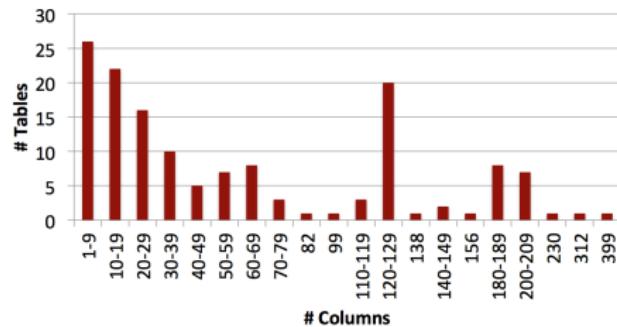
55% unused columns per company in average

40% unused columns across all companies



[Plattner, in-memory databases course]

Wide tables with unused columns



Column-oriented DB

Some issues we did not discuss:

- Indexes for the column store and buffer (ex: inverted index from value to positions).
- Parallelism (pipelines, data parallelism)
- Tuple reconstruction (early vs late materialization strategies).
- Logs

Outline

M2 D&K 2017

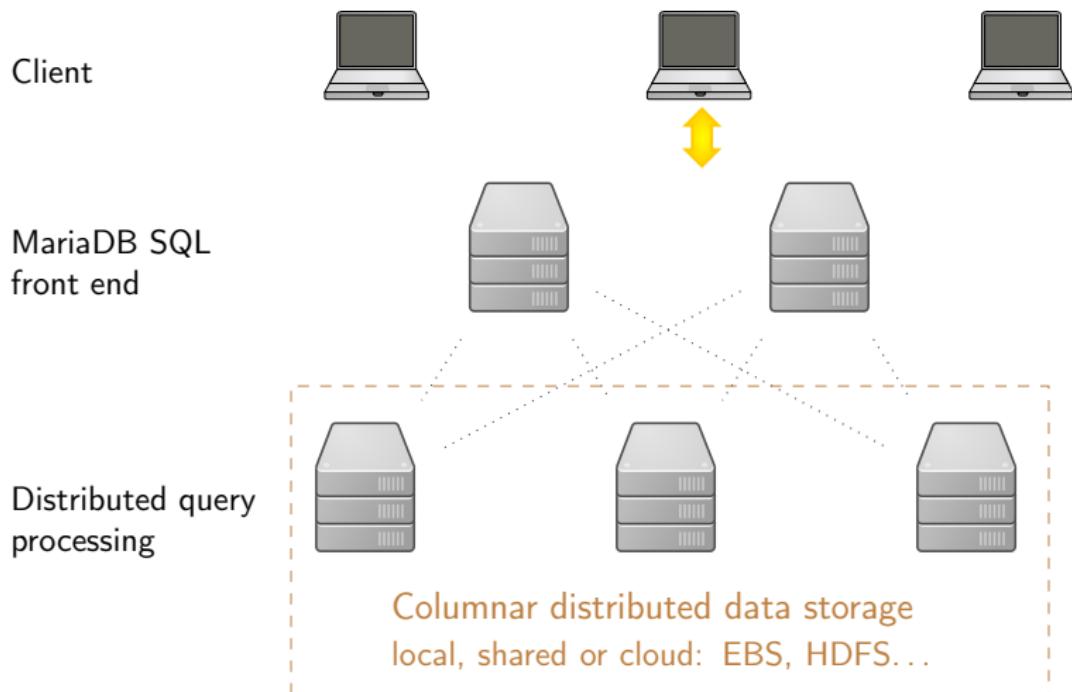
9 In-memory column stores

- Introduction to in-memory databases (insp. by H.Plattner's lectures)
- **In-memory databases: commercial systems: MariaDB**
- In-memory databases: commercial systems: Oracle
- In-memory databases: commercial systems: IBM DB2
- Techniques for handling updates

MariaDB ColumnStore

A massively parallel column-storage engine ported from InfiniDB.

Recommended over MariaDB row storage when queries process millions-billions rows from billions-trillions rows tables.



MariaDB ColumnStore: storage architecture

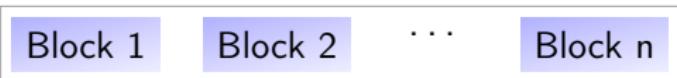
Partition



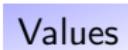
Segment file



Extent
8M values



Block
8kB of values



Values are fixed-length datatypes, 1-8bytes.
For larger values: pointer to dictionary entry.

MariaDB

Data is compressed using snappy library

≥250MB/s compression, 500MB/s decompression on single Core i7, 64bit.

Extent Map catalogs record the min and max value in each extent.
→ query engine prunes irrelevant extents.

Block-based MVCC for consistency.

A version buffer records in an in-memory hash table the blocks being modified by a transaction.

Dedicated cpimport bulk loader.

```
create table t (
  id int,
  Name varchar(20),
) engine=columnstore;
```



Outline

M2 D&K 2017

9 In-memory column stores

- Introduction to in-memory databases (insp. by H.Plattner's lectures)
- In-memory databases: commercial systems: MariaDB
- In-memory databases: commercial systems: Oracle
 - In-memory features
 - IM column store
- In-memory databases: commercial systems: IBM DB2
- Techniques for handling updates

Oracle 12.2: In-memory features

In-memory aggregation: a query transformation considered by query optimizer, like star transformation, materialized views, or expansion...

For each dimension:

1. compute a dense grouping key for rows satisfying filters
2. compute the vector of grouping keys on the fact table
3. build temporary table mapping grouping keys to attribute values

Then

1. Scan&aggregate the fact table using key vectors: VECTOR GROUP BY
2. join back dimension attributes using temporary tables.

In-memory column store: copy of tables, in column format (detailed later)

In-memory aggregation: example

```

SELECT category, country, state, SUM(amount)
FROM sales s, products p, geography g
WHERE s.g_id = g.geo_id
AND s.p_id = p.prod_id
AND g.state IN ('WA','CA')
AND p.manuf = 'ACME'
GROUP BY category, country, state
    
```

Geography

geo_id	city	state	country
2	Seattle	WA	USA
3	Spokane	WA	USA
7	SF	CA	USA
8	LA	CA	USA
...	France

Products

prod_id	category	manuf
4	sport	Acme
3	sport	Acme
1	food	Acme
8	electric	Acme
...	...	ATOS

dense	gr	key_g	state	country
1			WA	USA
2			CA	USA

dense	gr	key_p	category
1			sport
2			food
3			electric

temporary tables

Sales

p_id	g_id	amount
8	1	100
9	1	150
8	2	100
4	3	110
2	30	130
6	20	400
3	1	100
1	7	120
3	8	130
4	3	200

key vectors

dense	gr	key_p	dense	gr	key_g
3					
3			1		
1			1		
1					
2			2		
1			2		
1			1		

Oracle: IM column store

In-memory column stores. SGA pool that records copy of tables, in columnar format. Column store is recorded only in (volatile) memory.

Candidate for column store considered if declared in CREATE or ALTER statement. Once populated, kept consistent with the copy in row format.

```
ALTER TABLE t INMEMORY -- makes t candidate for populating the IM column store
  MEMCOMPRESS FOR CAPACITY HIGH
  PRIORITY LOW;
```

MEMCOMPRESS FOR

QUERY LOW : best for queries (default)

QUERY HIGH : higher compression

...

CAPACITY HIGH : highest compression

PRIORITY

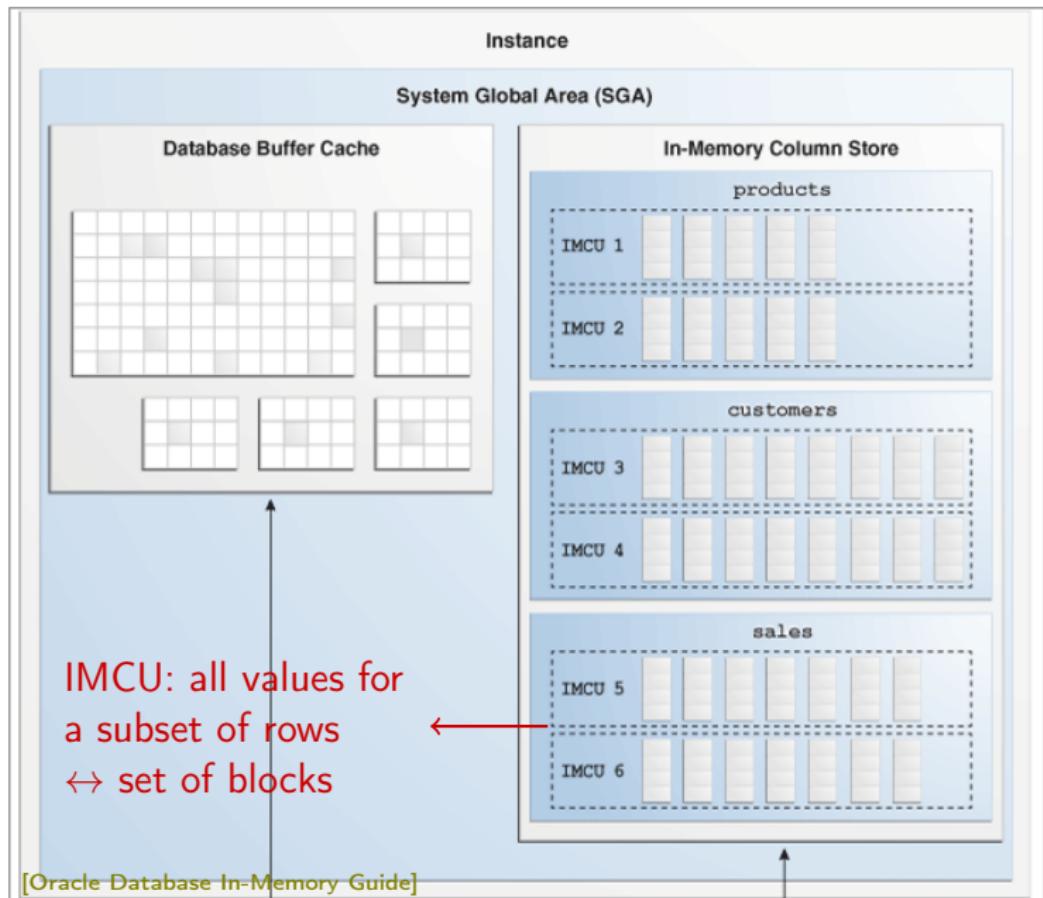
NONE : populates only when object is scanned (default)

LOW : populates after higher priority objects

...

CRITICAL : highest priority

Oracle: IM column store architecture



Oracle: IM column store architecture

For each IMCU= set of CU.

IMCU= In-Memory Compression Unit, CU= Column compression Unit

- header: metadata including
 - min/max value of each local column (*useful for pruning*)
 - (sometimes) local dictionary for local column data, implemented as a sorted list of distinct values with their dictionary code.
- body: the local column data, ordered by ROWID.

Products	
prod_id	category
4	2
3	2
1	1
8	0

IMCU 1

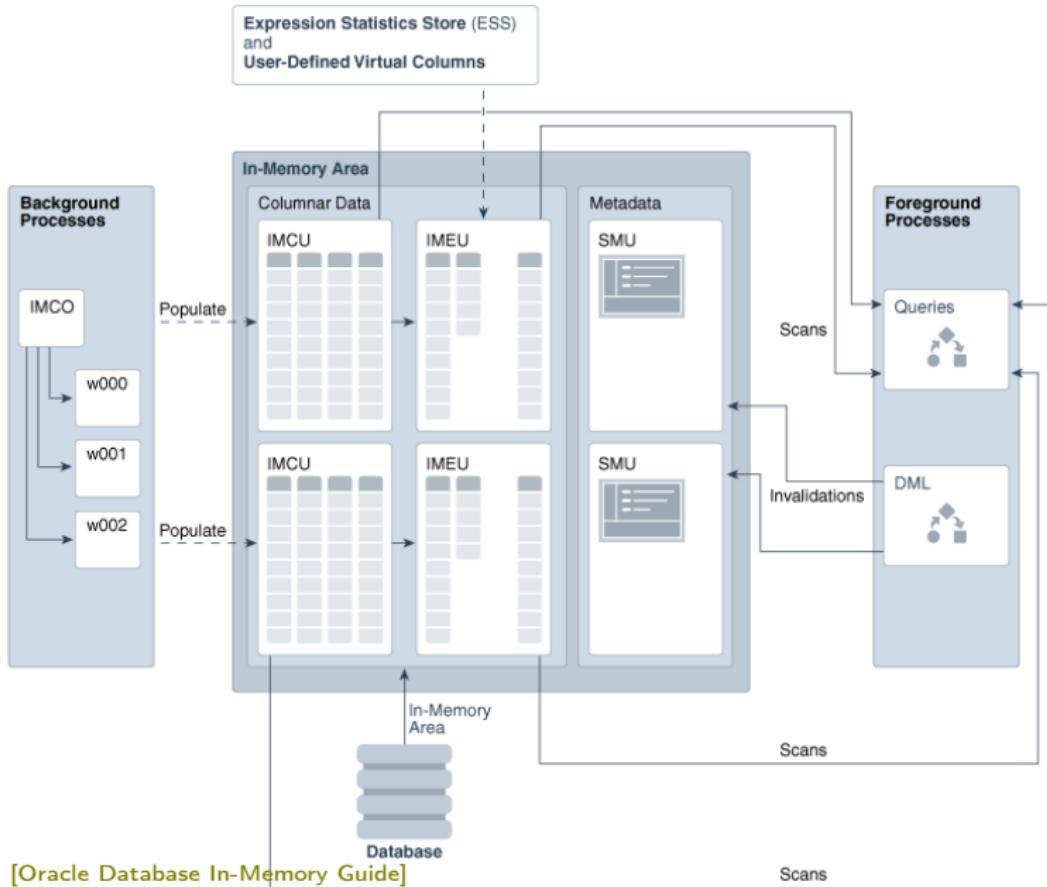
Value Code

min: Electric
max: Sport

Electric 0
Food 1
Sport 2

→ **min/max** allows IMCU pruning. Ex: *SELECT... WHERE prod_id > 9*

Oracle: IM column store architecture

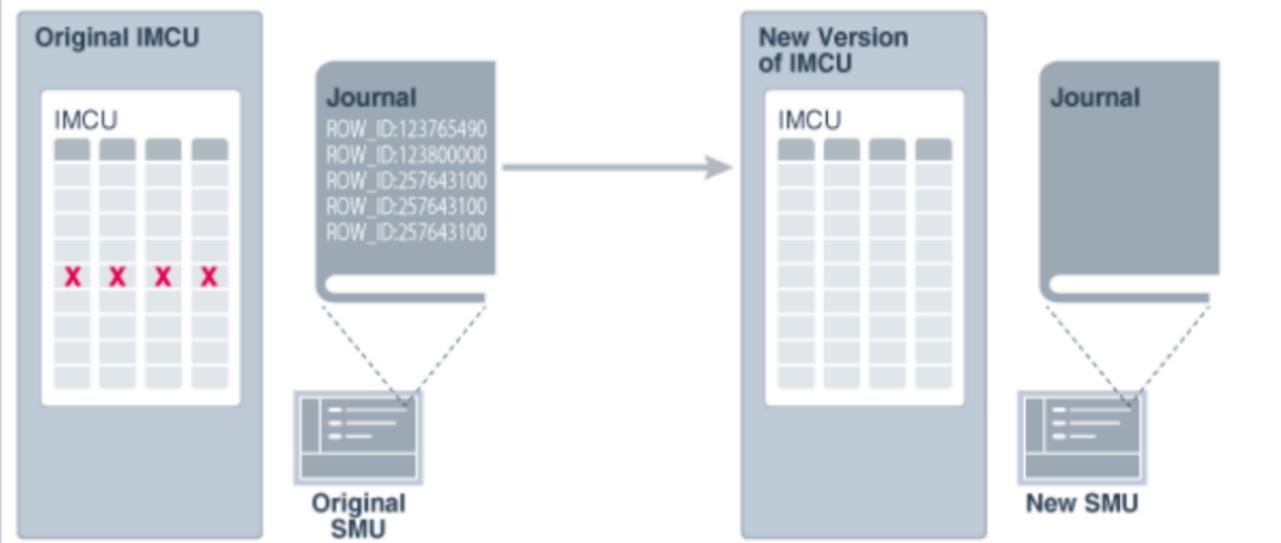


Oracle: IM column store updates

[Oracle Database In-Memory Guide]

In-Memory Area

- ① Mark original IMCU as old version as of SCN
- ② Create a new version of IMCU by combining clean rows from original with latest version of changed rows
- ③ Track any DML operations that occur during IMCU creation in Journal



- Threshold-based repopulation (number of changes in journal exceeds threshold)
- Trickle repopulation (periodic, updates all IMCU having some stale data)

Oracle: IM column store: integration on hardware

SQL-on-silicon: SPARC M7 chips:

SIMD on traditional CPU devised for graphics.

So devised chip with 8 Database accelerators with 4 pipelines = 32 engines supporting specialized instructions to process columnar data:

- Extract: uncompress data (byte/bit-packed, Huffman, RLE)
- Scan: filter data w.r.t an interval
- Select: filter data according to bit vector (given data vector and bit vector, return vector of selected items)
- Join

Some claims: 160GB/s bandwidth, 6x speedup on Apache Spark queries.

(Exadata) In-memory format in Exadata smart flash cache. Using additional flash cache to extend main memory (faster than disk): used to record db blocks evicted from SGA buffer cache.

Oracle vs SAP

SAP Hana: prototype around 2008, commercial product end 2011. Industry leader on in-memory techniques (though anteriority sometimes discussed)

2/3 of SAP Business suite customers rely on Oracle database.
By 2015, SAP tried to make ERP customers switch to SAP Hana:
integrated stack S/4HANA.

SAP pushing toward cloud-based apps.

2016: Oracle 12 integrates In-memory storage. Special care to support SAP BW.

In 2017, SAP signs with Oracle to support Oracle on their ERP till 2025.

[<http://www.silicon.fr/5-questions-comprendre-guerre-oracle-sap-in-memory-95002.html>]

[<http://www.scmfocus.com/saphana/2017/07/09/saps-change-policy-hana-oracle/>]

As long as we are comparing

Prix des options Oracle (21 juin 2016) « Processor Licence » :

Active Data Guard	11 500 \$
Database In-Memory	23 000 \$
Diagnostics Pack	7 500 \$
Tuning Pack	5 000 \$
Partitionning	11 500 \$
Advanced compression	11 500 \$
OLAP	23 000 \$
Advanced Analytics	23 000 \$
Spatial	17 500 \$
Multitenant	17 500 \$
TOTAL :	139 500 \$

[Source: <https://blog.developpez.com/sqlpro/p13001/ms-sql-server/>

`oracle-vs-sql-server-les-options-payantes-qui-font-la-difference]`

Oracle édition Enterprise : 47 500\$

Pricing of DBMS generally a bit opaque (multiple discounts, what should be counted: license cost only or ROI, etc).

Multiple discussions online about Oracle vs Microsoft SQL Server, MariaDB etc.

Outline

M2 D&K 2017

9 In-memory column stores

- Introduction to in-memory databases (insp. by H.Plattner's lectures)
- In-memory databases: commercial systems: MariaDB
- In-memory databases: commercial systems: Oracle
- In-memory databases: commercial systems: IBM DB2
 - In-memory features: DB2 12 for z/OS
 - In-memory Column store: DB2 BLU
- Techniques for handling updates

IBM DB2 12 for z/OS

DB2 12 (2016) emphasizes in-memory computations. Users must provision large RAM to benefit from enhancements.

In-memory contiguous buffer pool. Unlike versions < 12, makes sure page stealing only occurs in some 10% overflow area, which results in savings on cache page management.

In-memory index optimization. Reserves an in-memory area to speedup lookups in unique indexes.

Speeding up INSERTS. New insertion algorithm for inserts on non-clustered tables (table with MEMBER CLUSTER attribute). Requires more memory be assigned for table space partitions.

DB12 also kind of increases pool dedicated to sorting operations so they may fit in-memory...



IBM DB2 11.1 LUW with BLU acceleration

DB2 11.1 (2016), but BLU accelerations introduced in DB2 10.5 (2013).
BLU emphasizes processing compressed columnar data, parallelization,
SIMD, memory management...

For more, see: *DB2 with BLU acceleration*, VLDB'2013

<http://db.disi.unitn.eu/pages/VLDBProgram/pdf/industry/p773-barber.pdf>

```
CREATE TABLE Employee (
    ID SMALLINT NOT NULL,
    NAME VARCHAR(9),
    DEPT SMALLINT,
    SALARY DECIMAL(7,2)
)
ORGANIZE BY COLUMN;
```



References

Column/In-memory databases (general):

- *The Design and Implementation of Modern Column-Oriented Database Systems*,
Abadi et al. Foundations and trends in database, 2012

Column/In-memory databases (SAP):

- *In-Memory Data Management*, H.Platzner, livre disponible B.U. Orsay, et MOOC:
<https://open.hpi.de/courses/imdb2015>
- *Slides about SAP Hana (w. some numbers)*,
<http://web.stanford.edu/class/ee380/Abstracts/130522-slides.pdf>
- *Parallel Replication across Formats in SAP HANA for Scaling Out Mixed OLTP/OLAP Workloads*, VLDB'2017
<http://www.vldb.org/pvldb/vol10/p1598-han.pdf>
- *SAP HANA Adoption of Non-Volatile Memory*, VLDB'2017
<http://www.vldb.org/pvldb/vol10/p1754-andrei.pdf>

Column/In-memory databases (Oracle):

- <https://docs.oracle.com/database/122/INMEM/toc.htm>
- *Query Optimization in Oracle 12c Database In-Memory*, VLDB'15
<http://www.vldb.org/pvldb/vol8/p1770-das.pdf>
- *Distributed Architecture of Oracle Database In-memory*, VLDB'15
<http://www.vldb.org/pvldb/vol8/p1630-mukherjee.pdf>

References

Column/In-memory databases (MariaDB):

- <https://mariadb.com/kb/en/library/mariadb-columnstore/>
- <https://mariadb.com/kb/en/library/columnstore-storage-architecture/>

Column/In-memory databases (IBM):

- https://en.wikipedia.org/wiki/IBM_BLU_Acceleration
- *DB2 with BLU acceleration*, VLDB'2013
<http://db.disi.unitn.eu/pages/VLDBProgram/pdf/industry/p773-barber.pdf>
- <http://www.redbooks.ibm.com/redbooks/pdfs/sg248383.pdf>, other redbooks.

Trends in DWH:

- *Vendors white papers (Microsoft, Oracle, etc), Gartner, tdwi, etc. Ex:*
http://download.microsoft.com/download/C/2/D/C2D2D5FA-768A-49AD-8957-1A434C6C8126/The_Microsoft_Modern_Data_Warehouse_White_Paper.pdf

Data processing on modern Hardware:

- <http://www.odbms.org/wp-content/uploads/2014/03/Data-Processing-on-FPGAs.pdf>
- http://edbticdt2016.labri.fr/downloads/gustavo_alonso_slides.pdf

Outline

M2 D&K 2017

9 In-memory column stores

- Introduction to in-memory databases (insp. by H.Plattner's lectures)
- In-memory databases: commercial systems: MariaDB
- In-memory databases: commercial systems: Oracle
- In-memory databases: commercial systems: IBM DB2
- Techniques for handling updates

How storage architectures deal with updates



- ***update-in-place storage***: most relational DB. Limitations: versioning (SCD 2) results in fragmentation. Lock on page during update. Strategy often used with B-tree index. Provides optimal reads (≈ 1 seek per read/1 per scan if unfragmented). But updating index slower.
- ***log-structured storage***: all updates appended as a log (no “main”). Limitations: scans get expensive: must read all logs. But writes faster than in B-tree (and no lock). A popular version: *LSM-tree* (Log-Structured Merge tree).
- ***delta with main store***: main store is read-optimized, updates recorded in write-optimized buffer (*SAP Hana/Sans Souci, and other column stores*). Buffer merged from time to time.

→ **Rationale for log-structured storage: instead of writing a full page on each update, defer and process them in batch. Idea of *deferred writes* common with delta approach.**

LSM-tree

Storage optimized for write-throughput (as every log-structured storage):

- LSM-tree consists of 2 or more layers $C_0, C_1, C_2 \dots, C_k$.
- C_0 in-memory, can be tree, hash table...
- others (C_1, \dots) are B-trees, recorded on disk/slower memory.
- updates written only in C_0
- $\forall i$ when C_i is full, we merge it into C_{i+1} (C_i is emptied)
 - ↪ $\forall i$ versions in C_i always more recent than in C_{i+1}
 - ↪ at most r versions of record co-exist.
 - ↪ a Read must search layers $C_0, C_1 \dots$ until item found.

... but we did not detail what "full" means:

Original version:

[O'Neill et al. 1996]

- recommend using geometric progression: $|C_{i+1}| = r \times |C_i|$
 - ↪ total number of unique keys: $N = O(r^k)$.
- but claimed inefficient (does not exploit write locality)

References

- <http://www.vldb.org/pvldb/vol10/p1526-bocksrocker.pdf>
- <http://www.eecs.harvard.edu/~margo/cs165/papers/gp-lsm.pdf>
- <https://www.quora.com/How-does-the-Log-Structured-Merge-Tree-work>

Outline

M2 D&K 2017

10 Regular expressions

A useful tool: regular expressions

Usage:

Validate, search or replace text (in DW: filter&clean data).

Appear in:

- PHP, javascript (validate input, reformat), SQL (pattern matching)
- script languages/UNIX scripts: grep, sed, awk...
- perl
- programming language libraries: perl, python, java, c++...
- parsers, packet analysis...
- text editors/IDE (Find&Replace))

Caveat:

Several "standards", features vary slightly. Main flavours:

- POSIX flavours: Basic and Extended (BRE, ERE)
- PCRE (originally from Perl).

Regex Engines:

PCRE, Oniguruma, RE2 (Google), Boost (C++), RegExp (Javascript)...

regular expressions: memento:

<i>a</i>	symbol <i>a</i>
<i>(r)</i>	<i>r</i> (delimiter/capture)
<i>r₁ r₂</i>	<i>r₁</i> or <i>r₂</i> (alternative)
<i>r₁r₂</i>	concatenation

Special characters:

.	any symbol
^	text beginning
\$	end of text

Quantification:

<i>r?</i>	0 or 1 occurrence of <i>r</i>
<i>r*</i>	0 or more occurrences
<i>r+</i>	1 or more occurrences
<i>r{<i>n</i>}</i>	exactly <i>n</i> occ.
<i>r{<i>n</i>,}</i>	at least <i>n</i> occ.
<i>r{min, max}</i>	between <i>min</i> and <i>max</i> occ.

Captured subexpression:

<i>\n</i>	the substring matching <i>n</i> th captured group (defined by <i>n</i> th opening parenthesis)
-----------	-------------------------------------------------------------------------------------------------------------------------

Character classes:

<i>[a₁...a_n]</i>	1 character: <i>a₁</i> or <i>a₂</i> or...
<i>[a-d]</i>	<i>a, b, c</i> or <i>d</i>
<i>[^...]</i>	any character <i>but</i> ...

Predefined character classes:

<i>[:alpha:]</i>	<i>[A-Za-z]</i>
<i>[:alnum:]</i>	<i>[A-Za-z0-9]</i>
<i>[:space:]</i>	<i>[\t\r\n\v\f]=spaces</i>
<i>[:punct:]</i>	punctuation
<i>[:upper:]</i>	uppercase
<i>[:print:]</i>	<i>[\x20-\x7E]=visible char+space</i>

Predefined character classes (PCRE):

<i>\d</i>	decimal
<i>\h</i>	horizontal space character
<i>\s</i>	vertical space character
<i>\w</i>	word item

(uppercase to negate: *\D* = non-number.)

Metacharacters:

*^.[]\$()^+?|{} * escaped by \

Metacharacters for PCRE only:

!<>=:

regular expressions: examples

Special rules:

- by default, engine searches *first and longest* occurrence
- POSIX charact classes used within “[]”: ex: `[:alpha:]`
- predefined char classes determined by `LC_CTYPE` category in UNIX *locale*
- beware of digraphs, é may be 1 or 2 character, etc.
- *meta* status generally lost inside char class.
- when - first or last in “[]” : no interval but symbol - itself.
- `\0` captures whole string
- sometimes `$n` (outside pattern) instead `\n` (within) for backreference
- script/prog languages interpret pattern before forwarding to engine ⇒ escape symbols ex: `()\` ⇒ double escape!
- ... many options, vary a lot between tools.

Examples:

- `[a-z]+0` matches text containing one or more lowercase followed by 0.
- `^ [0-9]{10} $` matches text (line) that is a 10-digit number.
- `([a-c])z \1 \1` matches `azaa` but not `azcc`.

regular expressions: memento (advanced):

Misc.:

[= a=] equivalence class of "a" [àáâäãÃ...] defined in **LC_COLLATE** cat. (UNIX).

? : non-capturing group

\b word boundary (assertion, like ^ and \$)

Assertions : ! if negative, = if positive, < for lookbehind

(?=r) positive lookahead

(?! r) negative lookahead

(?<=r) positive lookbehind

(?<! r) negative lookbehind

Conditional pattern (only in some engines: python, perl, pcre)

(?(if) then | else)

Options (non-standard, but generally available under some form)

i case-insensitive

m multiline: if text has symbols, ^ \$ match line extremities

s single-line: enable "." to match newline char

x expanded: spaces ignored unless escaped...

Examples:

• new(?!s) over "Those news seem newer than new"

• (?ms)^a(.)*z\$ over "abcd\ngfz\na"

• regexp to validate password (≥ 8 symbols, digit, punctuation, uppercase) ?

regular expressions: greedy, lazy, possessive quantifiers:

 Greedy, lazy/reluctant, possessive quantifiers:

By default quantifiers are *greedy*: from a position, match as many occurrences as possible, then backtrack if no solution for global pattern.

- With `?` quantifier becomes *lazy*: the fewest occurrences, then increases if no solution.
- With `+` quantifier becomes *possessive* (Java, Python, Perl...): max occurrences, no backtracking even if it fails.

Examples:

- `ba*` over "abaaac"
- `ba*ac` over "abaaac"
- `ba*?` over "abaaac"
- `ba*?c` over "abaaac"
- `(ab{2,}+[a-z]` over "aabbc"
- `(ab{2,}+[a-z]` over "aabbb"
- `([a-c])*+cz` never matches.

Regular expressions: UNIX, python

- egrep = grep -E searches input file, returns lines where pattern found.

```
egrep 'ion$' /usr/dict/words
```

... returns words ending in *ion*.

- sed very powerful tool. Can modify text.

```
sed -e 's/before/after/g' infile.txt > outfile.txt
```

... replaces each occ. of before with after.

```
sed -i.back
```

```
's#([0-9]{4})-([0-9]{2})-([0-9]{2})#\3/\2/\1#g' a.txt
```

... changes date format inplace, backup (we escaped ()).

- python: functions match, search, findall, sub.

```
import re
c = re.search('(\d\d?) \w+ \d{4}', 'le 16 avril 2017')
print c.group(1) # 16
```

```
pattern = re.compile('\d\d? \w+ \d{4}')
```

```
c = pattern.search('le 16 avril 2017')
```

```
print c.group() # '16 avril 2017'
```

```
re.sub('([0-9]{4})-([0-9]{2})-([0-9]{2})','\1/\2/\3','2016-04-16')
```

```
# '2016/04/16'
```

Regular expressions: SQL

- Oracle : POSIX ERE compliant.

```
UPDATE countries
  SET name = REGEXP_REPLACE(name, '(.)', '\1 ') WHERE name != France;
-- name becomes: Brazil
SELECT first_name, last_name FROM employees
  WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$')
```

- PostgreSQL : implements regexp-like patterns with SIMILAR TO, some regexp POSIX functions:

```
SELECT col FROM t WHERE (col similar to '%(b|d)%');
-- returns "abc", but not "aca"
SELECT regexp_replace('foobarbaz', 'b..', 'X', 'g')
-- fooXX
```

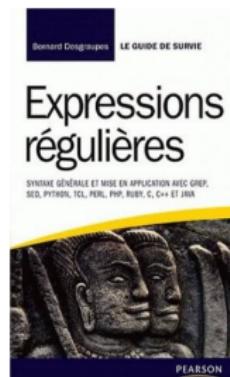
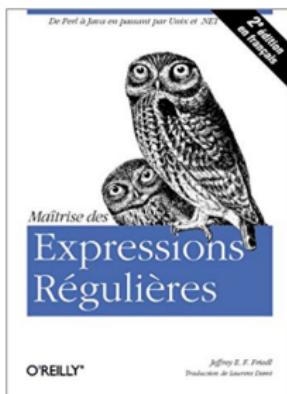
- MySQL : Herbert Spencer's regex library (POSIX)
- MariaDB : PCRE library (prev versions: regex)
- Microsoft SQL Server : partial support with LIKE (afaik)
- DB2 : no direct support (afaik) ⇒ UDF.

Can also call regexp library through UDF.

Bibliography:

- https://en.wikipedia.org/wiki/Regular_expression
- <http://www.regular-expressions.info>
- <http://www.expreg.com/presentation.php> (très complet)
- <https://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql/les-expressions-regulieres-partie-1-2>
- <https://stackoverflow.com/questions/22937618/reference-what-does-this-regex-mean>
- <https://regex101.com> (tester/débuger une regex)

À la B.U. (Paris-Sud) :



<http://www.regular-expressions.info/books.html> includes some book reviews.